INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia

de Segurança Informática

Vulnerabilidades nas conexões USB em Dispositivos com o Sistema Android

João Filipe Salgado Amarante

2017

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Mestrado em Engenharia

de Segurança Informática

Vulnerabilidades nas conexões USB em Dispositivos com o Sistema Android

João Filipe Salgado Amarante

Orientador:

Professor Doutor João Paulo Mestre Pinheiro Ramos e Barros

2017

Resumo

Vulnerabilidades nas conexões USB em Dispositivos com o Sistema Android

Nos últimos anos, a quantidade de ataques em *Smartphones* aumentou rapidamente, principalmente devido à complexidade de manter os Sistemas Operativos atuais a gerir esses dispositivos.

A complexidade de evitar vulnerabilidades nos sistemas operativos móveis atuais torna-os vulneráveis a muitos tipos de ataques. Esta dissertação apresenta informações resultantes do uso do *Android Debug Bridge* para extrair dados privados de *smartphones*. Foram identificados três cenários e foi desenvolvido uma prova de conceito. Ao ser executado num computador, o *script* é capaz de extrair dados privados de um *smartphone* quando este é conectado por USB. Em dois cenários foi possível extrair a informação de forma totalmente furtiva, sem o conhecimento do utilizador. No terceiro cenário, utilizando uma versão mais recente do Sistema Operativo *Android*, é necessária uma ação do utilizador, o que torna o ataque menos provável de ter êxito, mas ainda possível.

Palavras-chave: *Android*, Vulnerabilidade, USB, *Smartphone*, Dispositivo Móvel, Segurança Informática, Ataque físico, Internet das Coisas (IdC).

Abstract

Vulnerabilities in USB Connections on Devices with the Android System

In recent years, the amount of hacking attacks in Smartphones has increased rapidly, mainly because the complexity of maintaining modern operating systems managing these devices.

The complexity of avoiding vulnerabilities in the modern mobile operating systems makes them vulnerable to many types of attacks. This dissertation presents preliminary work resulting from the use of the Android Debug Bridge tool to extract private data from smartphones. Three scenarios were already identified and a concept proof script was developed. When running in a computer, the script is able to extract private data when a smartphone is connected by USB. In two scenarios it was possible to extract the information in a totally surreptitious way, without the user knowledge. In the third scenario, using a newer version of the Android operating system, a user action is needed which makes the attack less likely to succeed, but still possible.

Keywords: Android, Vulnerability, USB, Smartphone, Mobile Device, Computer Security, Physical Attack, *Internet of Things (IoT)*.

Agradecimentos

Inevitavelmente, as minhas primeiras palavras de agradecimento têm de ir, obrigatoriamente, para os meus pais. Sem o apoio, incentivo, paciência e compreensão, não teria siso possível concluir esta etapa na minha vida académica.

Agradeço ao Professor João Paulo Barros por toda a dedicação, empenho e apoio que me concedeu ao longo do desenvolvimento deste trabalho.

A todos os meus colegas e professores do Mestrado, por todo o companheirismo, disponibilidade e informações partilhadas, tornando este percurso mais fácil e enriquecedor, em especial ao Pedro Godinho, por todo o apoio, disponibilidade, dedicação e amizade ao longo deste período.

Por ultimo, mas não menos importante, à minha companheira de vida Núria Pires e ao meu filho Afonso que entretanto nasceu a meio do percurso por compreenderem, apoiarem e aceitarem a minha indisponibilidade durante largos períodos de tempo, dando sempre o seu contributo para que fosse possível terminar mais esta etapa da minha vida.

Índice

Índice de Figuras

Abreviaturas e Siglas

3DES	Triple Data Encryption Standard
3GP	Third Generation Partnership
AAA	Authentication, Authorization and Accounting
AAD	Advanced Audio Coding
ADB	Android Debug Bridge
AES	Advanced Encryption Standard
AIDL	Android Interface Definition Language
API	Application Programming Interface
APK	Android application package
ART	Android Runtime
ASLR	Address Space Layout Randomization
AVF	Android VPN Framework
AVRCP	Audio/Video Remote Control
B-BID	Battery-Based Intrusion Detection System
BSD	Berkeley Software Distribution
B-SIPS	Battery-Sensing Intrusion Protection System
CA	Certificate Authority
CDMA	Code Division Multiple Access
CFS	Completely Fair Scheduler
CPU	Central Processing Unit
DAC	Discretionary Access Control
DEX	Dalvik Executable
DMA	Direct Memory Access
DNS	Domain Name System
DoS	Denial Of Service
DPI	Dots Per Inch
DRAM	Dynamic Random-Access Memory
DSA	Digital Signature Algorithm

ECDSA	Elliptic Curve Digital Signature Algorithm
ES	Embedded Systems
EVDO	Evolution-Data Optimized
FDK	Font Development Kit
FLAC	Free Lossless Audio Codec
GCM	Google Cloud Messaging
GIF	Graphics Interchange Format
GNU	GNU's Not Unix
GPL	General Public License
GPRS	General Packet Radio Service
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HD	High Definition
HDR	High Dynamic Range
HID	Human Interface Device
HTC	High-Tech Computer
HTML	Hyper Text Markup Language
HTTPS	Hyper Text Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ID	IDentification
IDAMN	Intrusion Detection Architecture for Mobile Networks
IDS	Intrusion Detection System
IMA	Integrity Measurement Architecture
IOCTL	input/output control
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process Communication
IPS	Intrusion Prevention System
IPSec	IP Security Protocol
JNI	Java Native Interface
KDA	Keystroke Dynamic Authentication
L2TP	Layer 2 Tunnelling Protocol
LBA	Logical Block Addressing
LIDS	Linux Intrusion Detection System
LSM	Linux Security Modules
MAC	Mandatory Access Control
MAP	Message Access Bluetooth Profile

MD5	Message-Digest 5
MIDI	Musical Instrument Digital Interface
MIPS	Million Instructions per Second
ML	Machine Learning
MMS	Multimedia Messaging Service
MMU	Memory Management Unit
MP3	MPEG-1 and/or MPEG-2 Audio Layer III
MPEG	Moving Picture Experts Group
MSC	The mass storage class
MSD	Mass Storage Device
MTP	Media Transfer Protocol
NFC	Near Field Communication
NX	No eXecute
OHA	Open Hand-set Alliance
OMTP	Open Mobile Terminal Platform
OSSEC	Open Source Host-based Intrusion Detection System
OTA	Over-the-air
PC	Personal Computer
PIE	Position Independent Executable
PIN	Personal Identification Number
POSIX	Portable Operating System Interface
PPTP	Point-to-Point Transfer Protocol
QVGA	Quarter Video Graphics Array
RAM	Random Access Memory
ROM	Read Only Memory
RSA	Rivest-Shamir-Adleman Cryptosystem
RV/VR	Virtual Reality
RV4	Rivest Cipher 4
SD	Secure Digital
SDCC	Small Device C Compiler
SDK	Android Software Development Kit
SE	Standard Edition
SHA	Secure Hash Algorithm
Shmem	Symmetric Hierarchical Memory access
SIM	Subscriber Identification Module
SIO	Session Initiation Protocol
SMS	Short Message Service

SNI	Server Name Indication
SO/OS	Operating System
SoC	System-on-Chip
SQL	Structured Query Language
SSL	Secure Socket Layer
SVM	Support Vector Machines
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TPM	Trusted Platform Module
TUN	network TUNnel
UDP	User Datagram Protocol
UID	Unique Identification Code
UMTS	Universal Mobile Telecommunication System
URL	Uniform Resource Locator
USB	Universal Serial Bus
VM	Virtual Machine
VoIP	Voice over Internet Protocol
VP8	Video Codec Pack
VPN	Virtual Private Network
WebM	WebMedia Video File Format
WebP	WebPicture Image File Format
Wi-Fi	Wireless Fidelity
WVGA	Wide Video Graphics Array
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language
XSS	Cross-Site Scripting

Capítulo 1

Introdução

> Life's too short to remove USB safely...
>
> autor desconhecido

Num mundo cada vez mais dominado pelas tecnologias, a negligência no uso das mesmas oferece a oportunidade para cenários de ataque, que do ponto de vista dos programadores, dificilmente seriam equacionáveis. O presente trabalho é relevante para todos os utilizadores de dispositivos com o Sistema Operativo *Android* pois foca-se num tipo de ataque a estes dispositivos.

Os avanços recentes nas capacidades de *hardware* dos dispositivos moveis têm fomentado o desenvolvimento de Sistemas Operativos *open-source*. Esta nova geração de *smartphones*, onde se inclui o *iPhone* e os dispositivos com o *Google Android* são poderosos o suficiente para realizar a maioria das tarefas que anteriormente exigiam um computador pessoal.

Na verdade, este poder de computação recém-adquirido deu origem a muitíssimas aplicações que tentam alavancar o novo *hardware*: tarefas como navegação na Internet, *e-mail*, navegação GPS, mensagens e aplicações personalizadas, entre outras.

O uso omnipresente e a adoção generalizada do *Universal Serial Bus* (USB) [4] levou os fabricantes de dispositivos móveis a equipar a maioria dos telefones de terceira geração com portas USB.

1.1 Motivação e Âmbito do trabalho realizado

Entre os muitos dispositivos diferentes que fazem a IoT (Internet das Coisas), os dispositivos móveis são cada vez mais utilizados e representam um problema de

segurança crescente. Uma pequena falha, um erro, ou uma má interpretação duma entidade de segurança, pode talvez pôr em causa a segurança desses sistemas. Os ataques de proximidade são um bom exemplo disso, como por exemplo a segurança da conexão USB, que é muitas vezes negligenciada, podendo conter falhas menores mas que têm enorme impacto sobre a segurança do sistema. Este trabalho centra-se na segurança e privacidade em sistemas móveis, nomeadamente um tipo específico de ataque a esse tipo de dispositivo: a exploração de vulnerabilidades de ligação USB para obter dados privados em dispositivos *Android* [5].

O USB é atualmente utilizado principalmente como um meio de carregar a bateria, comunicação e sincronização dos conteúdos do telefone com computadores e outros telefones.

Sendo o *Android*, um Sistema Operativo que permite a terceiros desenvolver e contribuir com as suas aplicações, estão disponíveis um conjunto alargado de recursos que usam a interface USB para executar e sincronizar dados das mesmas.

Neste trabalho, procurou-se estudar as novas ameaças que decorrem do uso da interface USB para conectar e sincronizar o dispositivo móvel. Ao contrário das comunicações de rede e *bluetooth* para dispositivos móveis que têm mecanismos de defesa [6], o tráfego via USB não é autenticado, filtrado ou controlado, com exceção do *ADB Pairing*.

Para estabelecer conectividade *bluetooth*, o utilizador necessita introduzir uma palavra-passe para estabelecer a ligação entre dispositivos. Além disso, toda a comunicação móvel de pacotes e conexões sem fios são inspecionadas pela *firewall* ou sistemas de deteção de intrusão.

No caso do USB as conexões são ignoradas tanto pelos utilizadores como pelas defesas e são assumidos como um canal de comunicação confiável. Essa confiança inerente está enraizada na crença de que a proximidade física implica segurança/confiança.

Ataques de exploração de conexões via USB são algo incomuns e logo bastante ameaçadores. É nesse sentido que uma pesquisa contínua acerca deste tipo de ataques não deve ser negligenciada.

O *worm Stuxnet* [7] foi uma das ameaças mais importantes disseminadas através de *flash drives* USB. O *Stuxnet* apontou para computadores que executavam software que administrava sistemas de controle industrial em larga escala em grandes empresas de manufatura e de serviços públicos, explorando uma vulnerabilidade então não corrigida nos ficheiros de atalho do *Windows*.

Quando os utilizadores visualizavam o conteúdo de uma unidade USB infetada com um gestor de ficheiros como o *Windows Explorer*, o *Stuxnet* era carregado para

o computador. A Microsoft emitiu uma atualização de segurança *out-of-band* de emergência posteriormente para corrigir a vulnerabilidade no atalho.

O método de infeção USB já havia sido explorado anteriormente através do *worm Conficker* [8], que fez manchetes em todo o mundo depois que se espalhar utilizando também *flash drives* USB. Depois da aparição do *Conficker*, a *Microsoft* atualizou o *Windows* para corrigir um *bug* que impedia os utilizadores de desabilitar o "*AutoRun*", o mecanismo usado pelos *hackers* para infetar computadores automaticamente quando as unidades USB eram conectadas. A empresa também alterou o comportamento do *AutoRun* no Windows 7 para impedir tais ataques.

Ataques realizados através de USB, fornecem uma maneira muito eficaz de infetar grandes organizações, uma vez que essas organizações geralmente tendem a olhar mais para a defesa e contra ataques aos seus servidores e toda a infra-estrutura acessível *on-line*, esquecendo-se que os funcionários têm acesso dentro da infra-estrutura e podem muito bem compromete-la ao trazer uma *pen drive* USB já infetada. Os funcionários podem também intencionalmente conduzir atividades fraudulentas, e diretamente roubar informações importantes da empresa. Os funcionários podem utilizar o carregamento por USB através de um cabo conectado a um computador, simulando um simples carregamento de bateria de um *smartphone*, quando na realidade, estão utilizando essa conexão para transferir dados para o dispositivo.

Esta forma de obter informações importantes da empresa pode ser muito discreta, porque carregar a bateria do *smartphone* é uma atividade muito comum e que passa melhor despercebida.

Outra possibilidade é o facto dos funcionários serem completamente inconscientes do ataque. O transporte da *pen drive* USB de casa para a empresa ou qualquer outra localização intermédia, cria oportunidade para um potencial ataque indireto, como por exemplo no caso do *Stuxnet*, onde as casas, computadores pessoais e respetivas *firewalls* dos funcionários eram muito menos seguras e permitiram aos atacantes tirar proveito dessas circunstâncias.

1.2 Problema

A **Google** [9] disponibilizou dados que confirmaram que o *Android* está a ser utilizado por mais de 2,1 biliões de pessoas em todo o mundo. Com um número tão elevado de consumidores, o espaço para produzir um ataque malicioso é bastante alargado.

O problema consiste na possibilidade dos dados privados presentes num dispositivo com o Sistema Operativo *Android* poderem ser lidos através de uma conexão USB, sem conhecimento do dono dos mesmos.

Os ataques de proximidade, nomeadamente os baseados na conexão USB, são um bom exemplo disso e as evidências parecem indicar que os riscos associados são facilmente negligenciados pelos utilizadores. Na verdade, um comportamento semelhante já foi observado em relação às unidades USB [10].

1.3 Metodologia Utilizada

Nesta dissertação, foram exploradas vulnerabilidades no sistema operativo *Android* que são associadas às suas conexões via USB. Foram investigadas vulnerabilidades já documentadas na literatura e outras identificadas ao longo deste trabalho.

Em primeiro lugar, foi construida uma lista de ataques, seguidamente, elaborou-se um *script* que de acordo com o dispositivo conectado à porta USB, implementa os ataques correspondentes na lista.

Foram identificados três cenários e foi desenvolvido uma prova de conceito. Ao ser executado num computador, o *script* é capaz de extrair dados privados de um *smartphone* quando este é conectado por USB. Em dois cenários foi possível extrair a informação de forma totalmente furtiva, sem o conhecimento do utilizador. No terceiro cenário, utilizando uma versão mais recente do Sistema Operativo *Android*, é necessária uma ação do utilizador, o que torna o ataque menos provável de ter êxito, mas ainda possível.

1.4 Contribuições

O principal objetivo desta dissertação de mestrado foi o estudo detalhado e avaliação das vulnerabilidades que uma conexão USB apresenta para um dispositivo com Sistema Operativo *Android*.

Enumeraram-se os diversos mecanismos de segurança que permitem mitigar as vulnerabilidades avaliadas, com o objetivo de sensibilizar os utilizadores através da implementação de ataques bem sucedidos, para as boas regras de utilização em segurança do dispositivo.

Elaborou-se um *script* para obtenção de dados a partir de telemóveis quando conectados via USB a um computador.

Além das contribuições acima mencionadas, esta dissertação contém várias informações relevantes acerca do sistema Operativo *Android*, tais como a sua evolução

histórica, análises de segurança, o funcionamento do *Android Framework* e diversas formas de comprometer dispositivos com o referido SO.

1.5 Descrição do Documento

No resto do documento os capítulos estão organizados da seguinte forma:

Capítulo 2 Identifica as principais características do Sistema Operativo *Android* e as respetivas inovações inerentes a cada uma delas. Será também apresentado o modelo de segurança do *Android* tentando fornecer uma visão do estado corrente das vulnerabilidades encontradas.

Capítulo 3 Apresenta um estado da arte acerca do tema, através de uma revisão de literatura, que organiza cronologicamente o que anteriormente foi feito.

Capítulo 4 Apresenta os mecanismos de segurança atualmente existentes e descreve o seu funcionamento.

Capítulo 5 Define um cenário de ataque e a implementação do mesmo, bem como alguns resultados obtidos na exploração dessas vulnerabilidades.

Capítulo 6 Dá uma visão geral do trabalho desenvolvida, explicando o que foi realizado, dando também espaço para o trabalho futuro.

Apêndice Organização cronológica de todas as Versões do Sistema Operativo Android.

Capítulo 2

Sistema Operativo *Android*

O presente capítulo apresenta as principais características do sistema operativo Android dado ser este o objeto do presente trabalho.

Atualmente, o Sistema Operativo móvel da *Google* é o mais utilizado em todo o mundo, e está presente em milhares de aparelhos, de várias marcas. Mesmo com toda a dimensão que o sistema atingiu, a sua história é bastante recente. O primeiro dispositivo *Android* foi lançado em 2008 [11], há menos de 10 anos anos.

O Sistema Operativo *Android* surgiu em 2003, na cidade de Palo Alto na Califórnia e foi desenvolvido por Andy Rubin, Rich Miner, Nick Sears e Chris White, empresários já iniciados no ramo da tecnologia, que fundaram a *Android Inc.* Na ocasião, Rubin definiu a *Android Inc.* como: "Dispositivos móveis mais inteligentes e que estejam mais cientes das preferências e da localização do seu dono"[12]. No início a empresa desenvolvia todos os seus projetos de forma secreta e o objectivo era oferecer um Sistema Operativo *Open Source*, baseado no *Kernel Linux*.

Em 2005 a *Google* adquiriu a *Android Inc* e com isso nasceu a *Google Mobile Division*, divisão de pesquisa em tecnologia móvel da maior empresa do mundo de tecnologia. Apesar de ter causado desconfiança e dúvidas na época, já que muitos achavam difícil competir com o *Windows Mobile*, da *Microsoft*, e o iOS, da *Apple* [13].

Com a chegada do *Android*, o próprio conceito de *smartphone* foi remodelado. Na época, a *Google* teve uma ideia que se revelou extremamente importante para o desenvolvimento do sistema: oferecer 10 milhões de dólares aos programadores que conseguissem realizar as melhores aplicações para *Android* levando em consideração a primeira versão pública do *Android SDK*. As ideias enviadas pelos colaboradores ajudaram muito a criação da versão 1.0.

Hoje em dia o *Android* é o sistema mais utilizado no mundo com exceção do

Japão onde perde para o *iPhone* da *Apple* [14].

Todas as versões do *Android* foram denominadas por ordem alfabética e possuem nomes de doces. As exceções ficam por conta das versões 1.0 e 1.1, que não receberam nome, sendo chamadas de *Astro* e *Battenberg* pelos utilizadores. Depois vieram: *Cupcake*, *Donut*, *Eclair*, *Froyo*, *Gingerbread*, *Honeycomb*, *Ice Cream Sandwich*, *Jelly Bean*, *KitKat*, *Lollipop*, *Marshmallow* e *Nougat*. Além das versões oficiais e disponibilizadas ao público, existiram as versões Alpha e Beta.

2.1 O *Android Framework*

O *Android* é um ambiente de execução de aplicações para dispositivos móveis que inclui um sistema operativo, estrutura de aplicações (*framework*) e aplicações principais (*core*). As aplicações são escritas na linguagem de programação Java™ [15] com base nas APIs fornecidas pelo *Android Software Development Kit* (SDK). A base da *stack* de software *Android* é o *Kernel Linux* [16]. O *Android* usa o *Linux* [17] devido às *drivers* de dispositivos, gestão de memória , gestão de processos e redes. Normalmente, um programador de aplicações não programa diretamente nessa camada, sendo que o próximo nível contém as bibliotecas nativas *Android*. Estas bibliotecas são escritas em C/C ++ e são utilizadas por vários componentes do sistema nas camadas superiores. A Incorporação dessas bibliotecas em aplicações *Android* é conseguida através de interfaces de Java. Esta camada (*Native Libraries*) contém uma biblioteca de C personalizada, um motor de base de dados SQL, bibliotecas gráficas 2D e 3D, um motor de *browser* nativo (WebKit) e codecs (por exemplo, MPEG-4 e MP3).

O próximo nível é o *Runtime Android*, que consiste na máquina virtual (MV) Dalvik e nas bibliotecas do núcleo (*core*). A MV Dalvik executa ficheiros .DEX (executáveis Dalvik) que são mais compactos e eficientes para a memória do que os ficheiros produzidos pelo compilador Java. É um aspeto importante para dispositivos alimentados por bateria com memória limitada. As bibliotecas do núcleo são escritas em Java e fornecem um subconjunto substancial dos pacotes Java 5 SE (por exemplo, coleções padrão, I/O, rede, utilitários), bem como algumas bibliotecas específicas do *Android* que são necessárias para aceder aos recursos que o hardware, sistema operativo e bibliotecas nativas oferecem.

Atualmente a *Google* está a testar uma nova máquina virtual: a ART.

A camada *Android Framework*, ver Fig. 2.1, escrita inteiramente em Java™, inclui ferramentas fornecidas pela Google, bem como extensões proprietárias e/ou

serviços. Um componente importante do *Android Framework* é o *Activity Manager* (Gestor de actividade), que gere o ciclo de vida das aplicações.

A camada superior é a camada de Aplicações para a implementação de aplicações, tais como, telefone, *web-browser*, cliente de e-mail e muito mais.

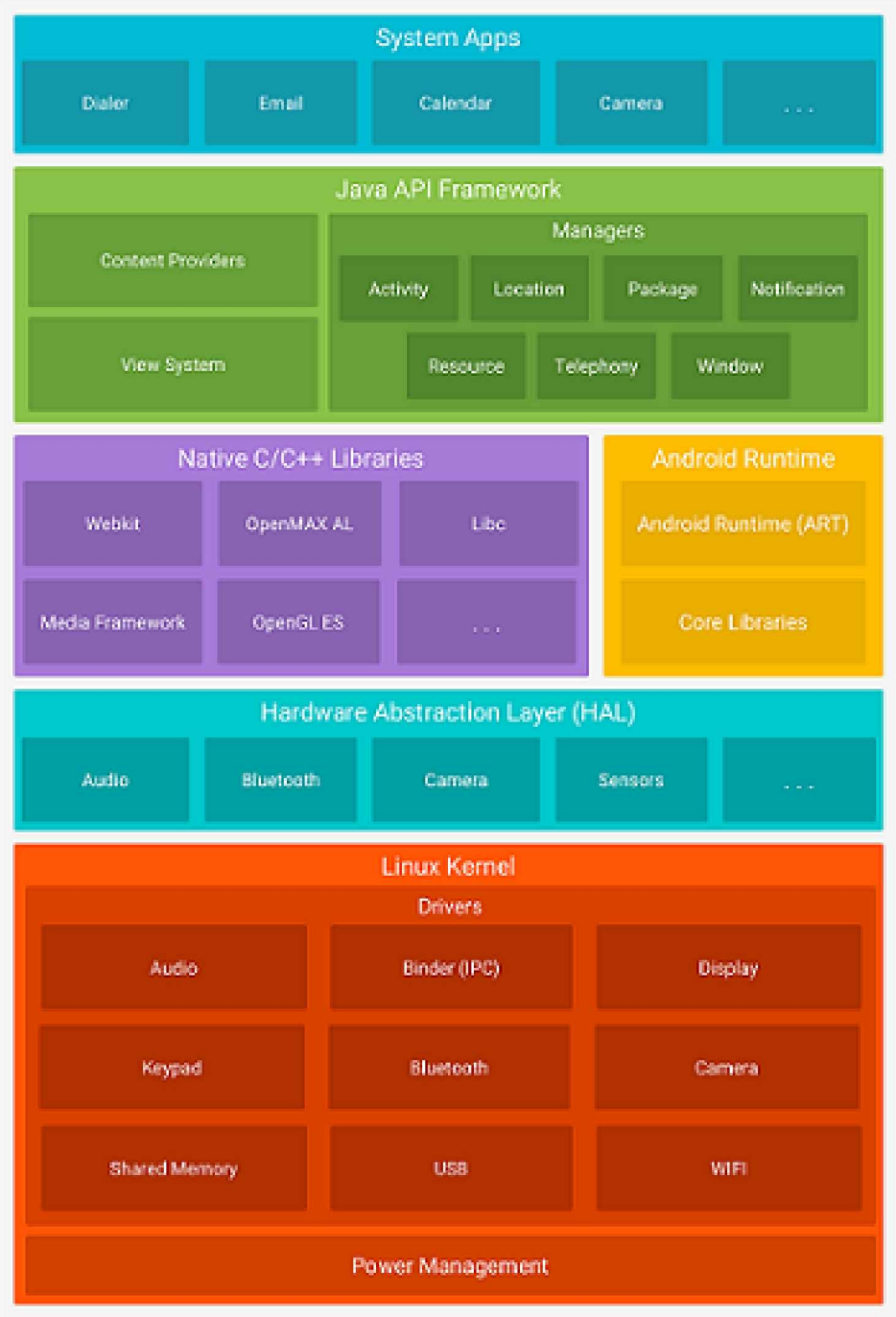

Figura 2.1: Android Framework (*in* [1])

Cada aplicação no *Android* é empacotada num .apk (*Android package*), ver Fig. 2.2, do ficheiro para instalação. O .apk é semelhante a um ficheiro jar Java padrão, que possui todos os recursos de código e não-código (por exemplo, imagens, *manifest*) para a aplicação.

O *Android package* é uma coleção de componentes. Esses componentes num .apk são isolados a partir de componentes noutro .apk e só podem comunicar uns com os outros e compartilhar dados através de meios fornecidos pelo sistema. Cada .apk está associado a um processo primário em que todos os componentes da aplicação (ou seja, atividades, serviços, recetores de radiodifusão e fornecedores de conteúdos) são executados. Esses componentes da aplicação, juntamente com os recursos, permissões e requisitos devem ser listados no ficheiro **AndroidManifest.xml**.

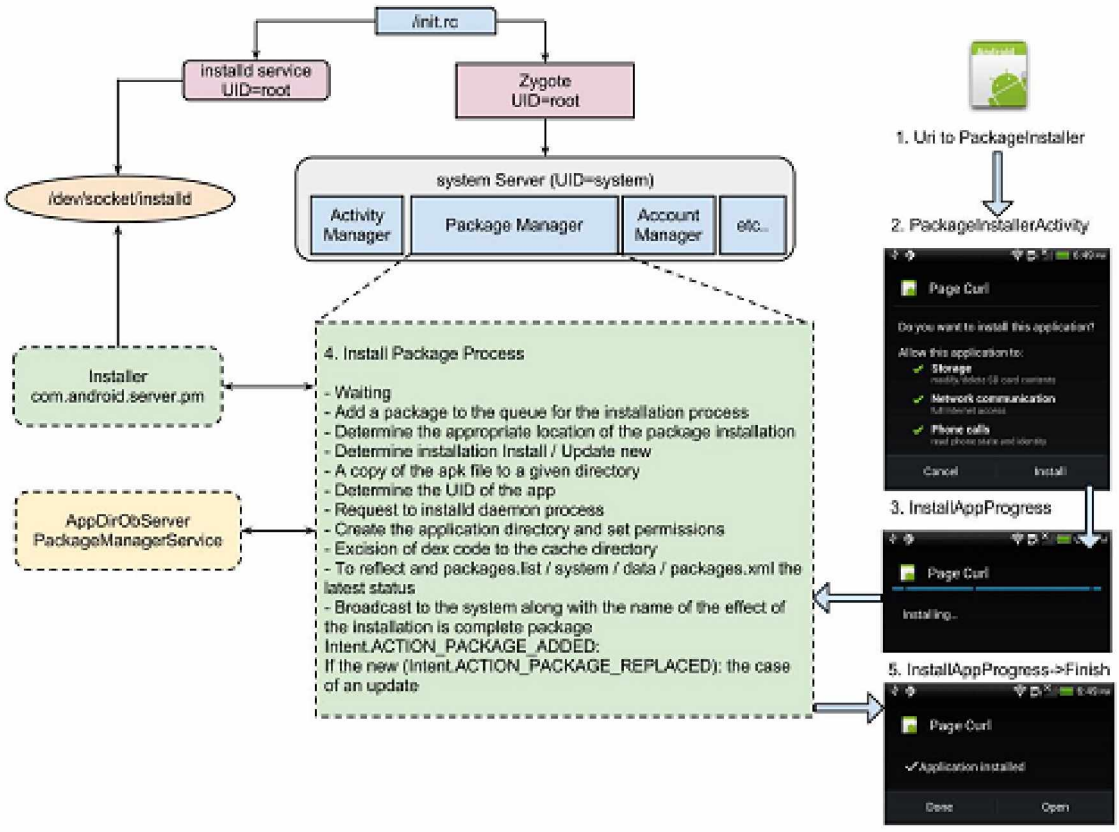

Figura 2.2: Android Package (*in* [2])

2.2 Mecanismos de Segurança do *Android*

O *Android* é um sistema multi-processo, onde cada aplicação (e partes do sistema) é executada no seu próprio processo. Na maioria dos casos, a segurança entre as aplicações e o sistema é aplicada ao nível do processo através de instalações padrão Linux, tais como IDs de utilizadores e grupos atribuídos a aplicações. Além disso, o controle de acesso é fornecido através de um mecanismo de permissão que impõe restrições sobre as operações específicas que uma determinada aplicação pode executar. As subsecções seguintes (2.2.1 a 2.2.8) apresentam diversos aspetos da segurança em *Android*. O texto baseia-se em diversas referências [18, 19].

2.2.1 Utilizadores POSIX (Portable Operating System Interface)

A cada ficheiro do *Android package* (.apk) instalado num dispositivo é dado o seu próprio e exclusivo Linux(POSIX) ID de utilizador. Esse ID de utilizador é atribuído quando a aplicação é instalada no dispositivo. De certa forma, é criada uma *sandbox* [20] que impede uma aplicação de entrar em contacto com outras aplicações. Para que duas aplicações possam compartilhar as mesmas permissões definidas e possivelmente serem executadas no mesmo processo, elas devem compartilhar o mesmo ID de utilizador, o que só é possível através do uso do recurso *sharedUserID*. Para que isto aconteça, as duas aplicações devem declarar explicitamente o uso do mesmo *sharedUserID* e ambas devem ter a mesma assinatura digital.

2.2.2 Acesso a Ficheiros

Os ficheiros no *Android* (de sistema e aplicações) são submetidos ao mecanismo de permissões do Linux, ou seja, cada ficheiro é associado aos IDs de utilizador (proprietário) e de grupo através de três tuplos de permissões, Ler, Escrever e Executar (rwx), sendo o primeiro tuplo para o proprietário, o segundo para utilizadores que pertencem ao grupo e a terceiro é para o resto dos utilizadores.

O Linux fornece muitas funcionalidades do sistema como por exemplo os *semifiles* [21]. Este mecanismo permite efetivamente a definição de permissões em ficheiros, diretorias, *drivers*, terminais, sensores de *hardware*, alterações de estado de alimentação, áudio, leituras de entrada direta, memória compartilhada e acesso a *backgrounds daemons*. Um aspeto que reforça estas várias medidas de segurança é que a imagem do sistema é montada apenas em modo de leitura (*read-only*).

Todos os executáveis importantes e ficheiros de configuração estão localizados ou no *ramdisk* (que é somente de leitura também, mas também reinicializada a partir de um estado conhecido em toda inicialização) ou na imagem do sistema. Portanto, um atacante que consiga gravar ficheiros em todos os lugares no *file system* vê ainda negada a possibilidade de substituir ficheiros críticos. No entanto, um atacante pode aumentar a complexidade do ataque e superar essa limitação, remontando a imagem do sistema, mas para isso, necessita acesso *root*.

Duas outras partições interessantes sobre o sistema de ficheiros do *Android* são a partição de dados e o cartão SD. A partição de dados é onde todos os dados do utilizador e aplicações são armazenados por defeito, podendo ser posteriormente transferidos para o cartão SD. Esta partição é distinta da partição do sistema, que efetivamente define uma quota na quantidade de dados de utilizador que podem ser introduzidos ou carregados no dispositivo. Esta quota efetiva impede a partição do sistema de danos no caso de o utilizador (ou um invasor mal-intencionado) instalar muitas aplicações ou criar muitos ficheiros. Quando o dispositivo *Android* é iniciado no "modo seguro", os ficheiros a partir da partição de dados não são carregados, o que torna possível a recuperação de tais ataques. No caso do cartão SD que é um dispositivo de armazenamento externo, os ficheiros podem ser manipulados fora do dispositivo e consequentemente fora do controle do mesmo.

2.2.3 Unidade de gestão de memória (MMU)

Um pré-requisito de muitos sistemas operativos modernos e Linux em particular, é a unidade de gestão de memória (MMU, de *Memory Management Unit*), um componente de *hardware* que facilita a separação de processos para diferentes espaços de endereço (memória virtual). Vários sistemas operativos utilizam o MMU de tal forma que um processo é incapaz de ler as páginas de memória de outro processo (divulgação de informações), ou de corromper a sua memória. A probabilidade de ordenação de privilégios é reduzida uma vez que um processo não é capaz de executar o seu próprio *run-code* num modo privilegiado, de forma a substituir a memória privada do SO.

2.2.4 Tipo de Segurança

O *Type safety* é uma propriedade das linguagens de programação. Obriga o conteúdo das variáveis a respeitar um formato específico e portanto, impede a utilização errada ou indesejável das mesmas. A ausência de *Type safety* ou de verificações de limite

podem levar à corrupção de memória e ataques de *buffer overflow*, que são meios para a execução de código arbitrário.

O *Android* utiliza Java, que é uma linguagem de programação orientada a objetos. Os programas escritos neste ambiente são menos suscetíveis a execução de código arbitrário. Esta particularidade está em contraste com outras linguagens tais como o C, que permitem *casting* sem a verificação do tipo e não realizam verificações de limite a menos que sejam especificamente escritas pelo programador. O *Android* permite que os programas tenham componentes nativos escritos em C, com o risco de segurança potencialmente reduzida. O *Binder*, mecanismo de comunicação entre processos (IPC) específico do *Android*, também é do tipo seguro. Os tipos de elementos de dados passados pelo *Binder* são definidos pelo programador no tempo de compilação no *Android Interface Definition Language* (AIDL). Isso garante que esses tipos são preservados através dos limites do processo.

2.2.5 Características de segurança de dispositivos móveis

Um conjunto básico de atributos dos sistemas de telecomunicações, vêm da necessidade de identificar o utilizador, monitorizar o uso e cobrar o cliente em conformidade. Um termo mais geral é o AAA (Autenticação, Autorização e Contabilidade). O *Android* como *smartphone*, utiliza esses recursos de segurança clássicos, sendo que a autenticação é geralmente feita por um cartão SIM e protocolos associados. O cartão SIM contém um código compartilhado apenas pelo cartão e o operador. Os principais mecanismos de segurança incorporados no *Android* [22] podem ser divididos em três tipos:

- Mecanismos Linux: **Utilizadores POSIX** onde para cada aplicação está associado um UID(*Unique Identification Code)* diferente impedindo-a de perturbar as outras e **Acesso a ficheiros** onde a diretoria da aplicação só está disponível se a mesma for proprietária impedindo-a de aceder a ficheiros de outra.

- Recursos de ambiente: **Unidade de gestão de memória** onde cada processo é executado no seu próprio espaço de endereçamento para prevenir *Privilege escalation*, divulgação de informações e DoS *Denial Of Service*, **Tipo de segurança** forçando a que o conteúdo variável adira a um formato específico, tanto em tempo de compilação como em tempo de execução prevenindo os *buffer-overflows* e o *stack smashing* e **Recursos de segurança do dispositivo móvel** utilizando o cartão SIM para autenticar e autorizar a identidade do utilizador evitando o roubo de chamadas telefónicas.

- Mecanismos específicos do *Android*: **Permissões de aplicação** onde cada aplicação declara que permissão requer no momento da instalação limitando as capacidades da mesma para evitar comportamentos maliciosos, **Encapsulamento de Componentes** onde cada componente de um aplicação (por exemplo, Atividade ou Serviço) possui um nível de visibilidade que regula o seu acesso a partir de outras aplicações (por exemplo, vinculação a um serviço) impedindo que uma aplicação perturbe outras, acedendo a componentes privados ou APIs, **Aplicações de assinatura digital** onde os ficheiros apk da aplicação são assinados pelo programador e verificados pelo gestor de pacotes efetuando uma correspondência e verificação de que duas aplicações são da mesma fonte e **VM Dalvik** onde cada aplicação é executada na sua própria máquina virtual evitando *buffer-overflows*, execução remota de código e *stack smashing*.

2.2.6 Permissões de Aplicações

O ponto fundamental da segurança a nível de aplicações no *Android* é o sistema de permissões que impõe restrições às operações específicas que uma aplicação pode executar. O *Package Manager* é responsável pela concessão de permissões para aplicações de instalação e a estrutura do mesmo é responsável por fazer cumprir as permissões do sistema em tempo de execução.

Existem muitas permissões embutidas em *Android* que controlam as operações, desde telefonar, tirar fotografias, utilizar a Internet, escrever uma SMS, etc. Qualquer aplicação *Android* pode declarar permissões adicionais. Para obter a permissão, uma aplicação deve explicitamente solicitá-la no seu manifesto ("contrato" da aplicação com o sistema *Android*).

As permissões possuem níveis de proteção associados: (1) Normal (permissões que não são especialmente perigosas), (2) Perigosas (permissões que são mais perigosas do que o normal, ou que não são normalmente necessárias às aplicações, tais permissões podem ser concedidas a uma aplicação com confirmação explícita do utilizador), (3) Assinatura (permissões que só podem ser concedidas a outros pacotes que são assinados com a mesma assinatura declarada na permissão) e (4) *SignatureOrSystem* (a permissão de assinatura que também é concedida aos pacotes instalados na imagem do sistema *Android*). A atribuição do nível de proteção é definida pelo programador e, no momento da instalação, as permissões solicitadas pela aplicação são concedidas com base em verificações das assinaturas das mesma que declaram essas permissões e interagem com o utilizador.

2.2.7 Encapsulamento de Componentes

Ao fornecer uma aplicação com a capacidade de encapsular os seus componentes (atividade, serviço, fornecedor de conteúdo e recetor de radiodifusão) dentro do conteúdo da mesma, o *Android* não permite qualquer acesso a eles a partir de outras aplicações (assumindo que eles têm uma *user-ID* diferente). Isto é feito principalmente definindo o recurso "*exported*"do componente se for definido como *false*, o componente em questão só pode ser acedido pela aplicação proprietária e qualquer outra aplicação que compartilhe o seu *user-ID*, se for definido como *true*, pode ser invocado por aplicações externas.

2.2.8 Pedidos de Assinatura

O sistema *Android* requer que todas as aplicações instaladas sejam assinadas digitalmente (recursos de código e não-código). O apk assinado é válido, desde que o seu certificado seja válido e a chave pública anexa verifique com êxito a assinatura.

2.3 Análise de Segurança

Os diversos aspetos da segurança do *Android Framework* podem ser agrupados e analisados em códigos de vários componentes *Android*, permissões de concessão de mecanismos e processo de instalação de aplicações e a aplicabilidade do Linux existente e *malwares* Java.

Nas subsecções seguintes, serão apresentadas diversas avaliações abrangendo os vários aspetos de segurança do *Android Framework* [23, 24].

2.3.1 Análise das camadas de *cornerstone do Android Framework*

As camadas de *cornerstone [25] do Android Framework* são as camadas pilares das bibliotecas *Android*, como por exemplo o **Android.app** que fornece acesso ao modelo de aplicação e é a pedra angular de todas as aplicações *Android*. Analisando as camadas inferiores do *Android*, é possível identificar três zonas críticas onde o código pode ser mais vulnerável e suscetível de conter *bugs* ou ser atacado: Linux Kernel, Bibliotecas de Sistema e Dalvik. Estas são seguidamente apresentadas.

Linux Kernel

O *Linux Kernel* tem algumas vulnerabilidades reconhecidas em termos de segurança, sendo os *drivers* [26] e as adições específicas do fornecedor dois dos locais que são particularmente "hospitaleiros"para *bugs*.

- **Submeter código na *mainline* [27]**, código que está integrado na *mainline* (que significa que estes núcleos são derivados (*branch*) do código de desenvolvimento principal do núcleo Linux) passa várias fases de verificações e validações. Quando o código é revisto, existe maior probabilidade de detetar *bugs* e alguns podem ter implicações de segurança. A este respeito, o *Android* divergiu do *Kernel* padrão, adicionando *drivers* e módulos específicos do fornecedor que não foram testados, sendo que alguns não são suscetíveis de ser incorporados sem um *redesign* completo.

- **Fase de desenvolvimento**, os programadores são obrigados a verificar se os *backdoors* das etapas de desenvolvimento e respetivos componentes de registo são removidos antes da implementação no ambiente de produção. Por exemplo, problemas na fase de depuração de código que envia para uma *root shell* cada tecla pressionada no teclado, permite a utilizadores experientes obter o controle total do sistema. Outro exemplo é o recurso de log do *Kernel*, que os programadores utilizam, mas que pode levar a uma negação de serviço pelos *too-verbose drivers* que não conseguem controlar o *rate-limit* das suas mensagens de *logging*.

- **Modificação e extensão de funcionalidades existentes**, algumas das modificações do Google são complementos a funcionalidades existentes. Nalguns casos, a melhor opção teria sido a integrar estas modificações noutros componentes. Um bom exemplo de tais componentes é o *Ashmem* que complementa o *Shmem*. O *Shmem* (Symmetric Hierarchical Memory access) é uma característica POSIX padrão que permite que vários processos possam partilhar a memória e o *Ashmem*, é um invólucro *Android* sobre o *Shmem* que utiliza a capacidade do *Kernel* para libertar as alocações de memória compartilhadas quando o sistema está "apertado"na memória. A decisão sobre qual o processo a matar é baseada em vários atributos, tais como a memória consumida, tempo de CPU, permissões etc.

- **Codificação de Utilizadores e grupos POSIX**, estas modificações contradizem as decisões do *design* de um projeto base em Linux. O *Android* sendo

uma plataforma de Linux usa grupos de IDs codificado para gerir a configuração de segurança sem extensões adicionais à infra-estrutura, aumentando a segurança, já que os serviços do sistema não são obrigados a correr com privilégios de *root*.

- **Configuração do *Kernel***, O *Kernel Linux* é altamente configurável e muitas opções comuns do Linux são "desligadas", a fim de reduzir o consumo de memória e implicitamente menos código significa uma redução na vulnerabilidade, por outro lado, isso também significa omitir módulos *security enabler* [28].

Bibliotecas de Sistema

O *Android* faz uso de muitas bibliotecas nativas, estas bibliotecas são destinadas a ser utilizadas quer por processos nativos, outras bibliotecas nativas ou pelo *Dalvik* através do *Java Native Interface*(JNI). O JNI é um método para chamar métodos nativos de Java no contexto do mesmo processo e é normalmente utilizado para fornecer funcionalidades de baixo nível, tais como, acesso *socket*, manipulação de ficheiros, criação de *threads* e comunicação entre processos internos. É também utilizado para implementar cálculos computacionalmente intensivos de multimédia, ocultar problemas de código e de licenciamento e aproveitar as bibliotecas existentes.

As bibliotecas nativas são escritas em C/C ++, que não é considerado seguro. Assim, as bibliotecas nativas têm uma maior chance de erros do que se fossem escritas em código Java. Desde que o JNI carrega as bibliotecas nativas no espaço de memória de um processo *Dalvik*, os erros na biblioteca nativa podem levar a falhas no processo do *Dalvik*, corrompendo a sua memória ou causar a execução de código arbitrário. Por esta razão, as bibliotecas do sistema são um alvo durante a pesquisa de vulnerabilidades de segurança.

Dalvik (tempo de execução)

O *Dalvik* é uma *Java Virtual Machine*(VM) com base no *Apache Harmony*, que foi amplamente modificada e adaptada para ambientes com pouca memória. O *Dalvik* oferece a possibilidade de executar código nativo através de JNI sem pedir permissão para isso. Proteger o *Dalvik* é crucial, uma vez que uma vulnerabilidade na VM afeta todas as aplicações. Um potencial ponto fraco é o *Dalvik VM .dex*, que contém o código de carregamento de ficheiros que é necessário para lidar com arquivos *.dex* a partir de fontes não confiáveis.

2.3.2 Permissões de nível de aplicação

Como o *Android* é um *open framework*, os dispositivos podem adquirir aplicações e serviços que os programadores desenvolvem. A desvantagem é que é difícil inspecionar e bloquear certas aplicações que não são confiáveis. O mecanismo de permissões [29] de nível de aplicação é responsável por assegurar que muitas das permissões do *core* são reservadas para aplicações da Google. No entanto, existe ainda espaço para outras permissões o que torna o dispositivo inevitavelmente vulnerável. Outra vulnerabilidade decorre do recurso *user-ID* compartilhado. Quando uma aplicação, que tem um *user-ID* compartilhado é instalada, todas as permissões concedidas são atribuídas ao utilizador que compartilha o *user-ID*. Uma generalização do cenário acima pode ser vista como duas aplicações aparentemente não relacionadas, que colaboram para obter informações a partir do dispositivo através de um meio compartilhado para transferir a informação entre as duas.

2.3.3 Instalar Aplicações

O *Package Manager* (serviço responsável pelo processo de instalação) valida a exatidão da *.apk* (*Android Package*). A validação inclui, (mas não está apenas limitada a) verificação da assinatura digital, confirmação da legitimidade das solicitações do *user-ID* ou permissões compartilhadas e a validação/verificação do ficheiro de *.dex* incluído. Devido à falta de uma autoridade de certificação (CA), não é possível verificar a identidade do programador, mas apenas a integridade da *.apk*. Como resultado, a menos que a aplicação use assinaturas relacionadas com recursos (permissões ou *user-ID* compartilhado) qualquer interferência com o *.apk* não será detetada.

Existem três métodos principais para instalar ficheiros *.apk* num dispositivo *Android*, diferindo apenas na forma como são obtidos e se a interação existe entre a aplicação de instalação do pacote (*Package Installer application*) ou o gestor de pacotes (*Package Manager*) diretamente.

- O *Android Debug Bridge* (ferramenta na linha de comandos que é fornecida juntamente com o SDK (*Software Development Kit*)) destinado a programadores, onde o comando de instalação é emitido a partir de uma máquina que está conectada a um dispositivo *Android* através de um cabo USB. Esta instalação é efetuada diretamente pelo gestor de pacotes sem qualquer interação do utilizador e permite uma automática concessão de permissões potencialmente perigosas uma vez que faz com que o processo possa ter um grande

impacto sobre a segurança do aparelho. Por conseguinte, este método não deve ser utilizado em aplicações em que o programador não é conhecido.

- O *Android Market* (Uma aplicação propriedade da Google que permite a navegação e *download* de aplicações que foram publicadas por diferentes programadores).

- Nenhuma das anteriores (por exemplo, a partir do cartão SD); estas, ao contrário das aplicações obtidas a partir do *Android Market*, requerem uma confirmação do utilizador para que fique consciente dos riscos envolvidos na instalação de aplicações de fontes desconhecidas

Por padrão, é proibida a instalação de aplicações de fontes desconhecidas. Note-se que esta restrição é imposta pelo *Package Installer application* e não pelo *Package Manager*, por isso, não se aplica a métodos que não utilizam o primeiro e usam o segundo diretamente. Uma falha de *design* no *Package Installer application* permite a atacantes obter silenciosamente permissões ilimitadas em dispositivos comprometidos [30].

2.3.4 *Web-Browser*

Navegar na *Web* expõe os utilizadores do *Android* a ataques comuns, tais como: *Cross-Site Scripting* (XSS); ataques de codificação de URL; Engenharia social; e *scripts* maliciosos. *WebKit*, o motor *Web open-source* do *Android*, tem uma larga história de vulnerabilidades [31]. Alguns ataques recentes no *browser* incluem *buffer overflow* numa biblioteca nativa já ultrapassada e uma vulnerabilidade de XSS explícito [32][33]. Ambos os ataques permitem que o invasor execute qualquer código malicioso no dispositivo com todas as capacidades e privilégios atribuídos ao navegador *Web*.

2.3.5 *SQL Injection*

O *SQLite* é o provedor de armazenamento persistente mais comum na plataforma *Android*, é usado pela vasta maioria das aplicações de sistema e de utilizador. A API do *Android* SDK trabalha automaticamente ou manualmente com bases de dados *SQLite* com respeito a incorporação de dados inseridos pelo utilizador em consultas. A incorporação manual é executada anexando os parâmetros para uma cadeia de consulta SQL. Este método é suscetível de injeção SQL. Uma alternativa é ligar automaticamente os valores de parâmetros de uma maneira segura.

2.3.6 Conectividade e comunicação

Vários meios de transporte de comunicação (Bluetooth, Wi-Fi, GPRS, UMTS, cabo) oferecem várias opções de *malware* para infiltrar um dispositivo. Alguns *malwares* podem propagar-se através de diferentes meios de transporte. O trabalho realizado e descrito nesta dissertação enquadra-se neste tema: a exploração de vulnerabilidades na conexão USB.

2.3.7 *Hardware*

O *Hardware* de um dispositivo *Android* também pode ser atacado, uma vez que, vários componentes são particularmente vulneráveis. Tais casos são aplicáveis principalmente devido à tendência do *Android* para permitir que as aplicações possam aceder à plataforma tanto quanto possível.

O armazenamento flash (cartão SD, interno e do cartão SIM) pode ser desgastado uma vez que têm número finito de ciclos de apagar e o *Android* não avalia o limite de I/O ou permite quaisquer quotas totais de I/O. Drenar a bateria é trivial (por exemplo, mantendo o CPU a correr ou pela realização de um *wake-lock*) [34].

2.3.8 Atualizações de *Software*

A atualização de *Software* é um mecanismo de segurança usado de forma comum por todos os utilizadores, que fornece a capacidade de atualizar o sistema com correções para lidar com vulnerabilidades ultimamente descobertas. Quando conectado à *Internet*, o sistema *Android* pode ser forçado com atualizações de *Software* para corrigir problemas conhecidos em dispositivos vulneráveis.

O ficheiro de atualização, assinado pelo fornecedor é verificado em duas etapas: em primeiro lugar através de *download* utilizando chaves públicas que vêm com o dispositivo e, em segundo lugar, através do utilitário de recuperação utilizando chaves públicas adicionais que são codificados para o executável. O sistema não irá instalar uma atualização que não esteja assinada por uma chave aprovada. Por conseguinte, conclui-se que o *design* do mecanismo de atualização de *software* "escuta"e assume que a aplicação não tem erros, não sendo possível instalar uma atualização que não foi criada pelo proprietário das chaves de atualização.

Até agora, o *Android* teve várias atualizações que incluem correções para lidar com erros, redução da vulnerabilidade e para a adição de recursos adicionais. Uma das principais vulnerabilidades existia na *root shell*, onde foi usado o primeiro método *jail-breaking* de dispositivos *Android*. Ao ganhar acesso de *root* no aparelho, um *jail-*

breaking é capaz de prevenir quaisquer futuras atualizações, utilizando conhecidos ataques de elevação de privilégios locais, ou provocando *downgrades* de *firmware*. O *downgrade* de *firmware* é um método que é utilizado para obter privilégios de *root* no aparelho. Se uma versão de *firmware* anterior é conhecida por ser vulnerável e fornece um método para a obtenção de privilégios de *root*, o utilizador/atacante vai tentar rebaixar a versão do *firmware* para a versão vulnerável.

2.3.9 Relevância de Linux *Malware* Existentes

Um atacante pode tentar atingir os serviços, "escutando"portas de *host* locais que seriam inacessíveis externamente. Estes podem incluir serviços *Android*, como o *dbus* (Processos internos de comunicação *Bluetooth*) ou *mountd* (manipulação de sistemas de ficheiros de montagem). Uma vez que alguns destes serviços são executados com privilégios de *root*, qualquer *bug* explorável encontrado relacionado com eles poderia permitir que um invasor execute o seu código, juntamente com privilégios de *root*.

O código do *kernel* subjacente é também uma fonte de erros exploráveis. Qualquer vulnerabilidade de uma versão do *kernel* prévia ou atual do *Android* pode ser obviamente explorável no mesmo. As questões de segurança do código do *kernel* são suscetíveis de ser mais difíceis de resolver devido ao atraso de tempo envolvido com a emissão de um grande *patch* para o *kernel* de uma ampla base de utilizadores. Este intervalo de tempo pode deixar dispositivos vulneráveis por longos períodos.

2.3.10 Relevância de Java *Malware* Existentes

A plataforma *Android* não suporta *applets* Java [35] (devido a problemas de licenciamento), ou *Adobe Flash* [36] e portanto, está imune à maioria dos *exploits* Java baseados na *web* para evitar a exposição potencial do sistema a ataques específicos de *Dalvik* através da mesma.

Os vírus Java não são aplicáveis à plataforma *Android* por duas razões. Em primeiro lugar, infetam formatos de ficheiros de classe e devem ser ajustados para suportar a injeção de código malicioso em ficheiros *Android* .DEX binários. Em segundo lugar, no *Android*, as aplicações não têm privilégios de gravação em qualquer pacote de ficheiros (.apk), inclusive na própria aplicação.

2.4 Avaliação de Segurança do *Android Framework*

Um dispositivo *Android* no seu estado normal é seguro uma vez que nem os componentes do núcleo nem o *kernel* podem ser substituídos por um atacante (ou pelo proprietário legítimo), a menos que o *hardware* tenha sido manipulado, o que é difícil de executar. A única maneira de alterar componentes do sistema operativo é identificar uma vulnerabilidade num dos módulos do *kernel* ou bibliotecas centrais que permitam ao *malware* adquirir acesso *root* [28].

Sempre que é difundido um *bug* ou vulnerabilidade num dos principais componentes (tais como uma biblioteca nativa ou um componente do *kernel*), um invasor pode ser capaz de executar código malicioso num modo altamente privilegiado e até mesmo ganhar controlo total sobre o dispositivo. Esta ameaça é ampliada devido ao fato do código-fonte do *Android* ser acessível ao público, alguns processos do sistema serem executados com privilégios de *root* e não existir nenhum mecanismo de controlo de acesso "refinado" para processos do sistema.

Tornar o código fonte disponível ao público proporciona certos benefícios, entre eles, muitos indivíduos serão capazes de controlar e verificar o código. Mesmo que a plataforma nunca seja totalmente segura, a abordagem de código aberto promove constantes melhorias de segurança.

Para atacar remotamente um dispositivo *Android*, é necessário que o dispositivo tenha exposto um serviço vulnerável na rede. Esta exigência reduz a probabilidade de cenários de ataque remotos já que nenhum dos padrões dos serviços está "à escuta" de conexões. A quantidade de código explorável a correr num dispositivo através de serviços locais, *drivers* de dispositivo e código do *kernel* torna a exploração *host-based* numa operação de risco mais elevado.

No seguinte capítulo será apresentado o estado da arte, bem como diversos trabalhos relacionados.

Capítulo 3

Estado da arte

Neste capítulo, é apresentado o resultado de uma investigação sobre a proteção de dispositivos móveis, bem como vulnerabilidades exploradas recentemente.

Entre os sistemas operativos de *smartphones* mais significativos que surgiram recentemente encontra-se o *Google's Android framework*. O desafio para garantir a segurança de um *Android* é cada vez mais semelhante à de um computador pessoal. Os fabricantes de SO de dispositivos móveis, estão agora a enfrentar os mesmos problemas e desafios de segurança que os fabricantes de PC se debateram ao longo dos anos. O número crescente de ataques a plataformas móveis (Especialmente em *smartphones*), juntamente com o seu uso crescente tem levado muitos fornecedores e investigadores a propor uma variedade de soluções de segurança. Por exemplo, a **Symbian** e a **Google** projetaram os seus sistemas operativos para ativar aplicações que são executadas apenas em *sand boxes* especializadas, minimizando a capacidade do *malware* se espalhar como é possível observar na Fig. 3.1 [37].

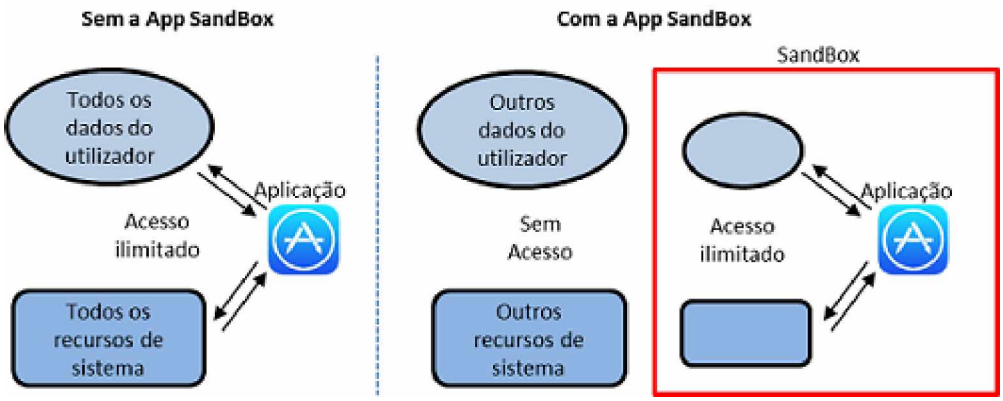

Figura 3.1: Exemplo de uma aplicação com e sem *Sandbox*

3.1 Trabalho Relacionado

A conexão USB é apenas uma das várias maneiras de obter dados de dispositivos móveis. Um catálogo desses métodos, que fornece um bom ponto de partida para este assunto, está já disponível [38].

Depois de efetuar uma revisão de literatura, foi possível apurar que a maioria das pesquisas existentes sobre a proteção dos dispositivos de comunicação móvel, têm-se centrado nos sistemas de intrusão *host-based*. Estes sistemas, que utilizam a deteção de anomalias ou métodos baseados em regras, extraem e analisam (localmente ou por um acesso remoto), um conjunto de características que indica o estado do dispositivo.

A investigação sobre a proteção dos dispositivos móveis pode ser assim organizada cronologicamente:

1. Em 1997, as Redes Neuronais Artificiais foram usadas por Moreau [39] com a finalidade de detetar comportamentos anómalos indicando uma utilização fraudulenta dos serviços de telefonista (ex: Registo com uma identidade falsa e usar o telefone para destinos de tarifas elevadas) com base em 16 características que representam média e desvio padrão da duração total e número de chamadas nacionais e internacionais de longo/curto prazo.

2. Também em 1997, o sistema *Intrusion Detection Architecture for Mobile Networks* (IDAMN) de Samfat [40], usa os dois métodos de deteção baseado em regras e anomalias. O IDAMN oferece três níveis de deteção: deteção baseada em localização (um utilizador está ativo em dois locais diferentes ao mesmo tempo); deteção de anomalias de tráfego (Uma área que tem a atividade da rede normalmente baixa regista a atividade de rede altas) e deteção de comportamento anómalo de utilizadores de telemóveis individuais. Com a finalidade de detetar comportamentos anómalos, um perfil é gerado pela monitorização da atividade de telefone do utilizador (a duração da chamada, o tempo de inatividade entre duas chamadas, número de transferências realizadas, o número total de chamadas durante uma sessão e duração total de chamadas durante uma sessão). Além disso, a localização do utilizador na rede (*roaming*) é monitorizada através da criação de uma máquina de estado com a probabilidade de se deslocar de um local (celular) para outro.

3. Em 2002 Sheyner et al. [41] descreveram outro tipo de ataques contra os cartões SIM de GSM/GPRS que são os ataques *side-channel* que permitem a um atacante obter informações sensíveis a partir de outros *side-channel*.

4. Em 2004 Xu e Zhu [42] estudaram a possibilidade de lançar ataques e *spam* através de *Trojan* em aplicações instaladas pelo serviço de notificação de abuso personalizado. Os resultados experimentais foram apresentados e os autores apresentaram uma abordagem para o fornecimento de conteúdo de *spam* furtivo que pode ajudar na identificação da aplicação/*Trojan* que consegue ultrapassar o processo de revisão da *Google Play Store*.

5. Em 2005, Yap et al. [43] empregam uma solução verificadora de comportamentos que podem detetar atividades maliciosas no sistema. É apresentado um cenário de prova-de-conceito que usa um telemóvel Nokia executando um sistema operativo *Symbian*. Na demonstração, um detetor de comportamentos, deteta um *Trojan* a tentar usar o componente de servidor de mensagens sem autorização para criar uma mensagem SMS.

6. Também em 2005, Nash et al. [44] apresentaram um projeto para um sistema de deteção de intrusão que estima o consumo de energia de acordo com um modelo de regressão linear, com base em parâmetros tais como acessos de carga do CPU e do disco, para determinar a quantidade de energia utilizada numa base por processo e identificar processos que podem consumir potencialmente baterias em excesso.

7. Em 2006, Jacoby e Davis [45] apresentam uma série *Battery-Based Intrusion Detection System* (B-BID). A ideia base é que a monitorização da corrente elétrica e avaliação da correlação do dispositivo com as assinaturas e padrões conhecidos podem facilitar a deteção de ataques.

8. Também em 2006, Miettinen et al. [46] alegaram que as abordagens baseadas em *hosts* são necessárias, uma vez que a monitorização baseada em redes por si só não é suficiente para encontrar as ameaças futuras, ou seja, deve ser adotada a coleta de dados baseada em *hosts* e em redes, com a finalidade de ser capaz de utilizar os pontos fortes de ambas as abordagens de deteção. Um mecanismo de correlação no servidor de *back-end*, filtra os alarmes recebidos de acordo com regras de correlação na sua base de conhecimento e encaminha os resultados da mesma para uma interface gráfica de monitorização de segurança que será analisada pelos administradores de segurança.

9. Em 2007, Cheng et al. [47] apresentam o *SmartSiren*. Um sistema de deteção de vírus *proxy-based*. Dispositivo únicos e comportamentos anormais de todo o sistema são detetados pela análise conjunta da atividade de comunicação

de *smartphones* monitorizados. A arquitetura *SmartSiren* consiste num *proxy* de *back-end* que interage com agentes sobre os dispositivos protegidos. Os agentes simplesmente recolhem informações e retransmitem para o *proxy* que realiza a análise e envia os alertas.

10. Em 2008, um quadro de deteção de comportamento interessante é proposto por Bose [48] para detetar *mobile worms*, vírus e cavalos de Troia. O método emprega uma abordagem lógica temporal para detectar atividades maliciosas ao longo do tempo. Um comportamento de *malware* é representado por descrever a ordenação temporal de ações de uma aplicação que pode revelar intenções maliciosas, mesmo quando cada ação por si só pode parecer inofensiva. Uma base de dados de assinaturas de comportamento malicioso foi gerada para estudar mais de vinte e cinco famílias distintas de *Symbian OS malware*. De seguida, uma técnica de mapeamento de duas fases constrói estas assinaturas em tempo de execução dos eventos do sistema monitorizados e chamadas de *API do Symbian OS*. O sistema diferencia o comportamento malicioso de *malware* a partir do comportamento de aplicativos benignos através da formação de um classificador baseado em *Support Vector Machines* (SVM).

11. Também em 2008, Kim [49] apresenta um *framework power-aware* de deteção de *malwares* que monitoriza, deteta e analisa previamente ameaças desconhecidas de exaustão de energia. O quadro *framework* recolhe amostras de consumo de energia e gera assinaturas, estas assinaturas são usadas para classificar o *malware* móvel, medindo a semelhança entre as assinaturas de energia usando a medida *x2-distance* .

12. Ainda em 2008, o *Battery-Sensing Intrusion Protection System* (B-SIPS) criado por Buennemeyer et al. [50] para computadores portáteis, alerta quando alterações anormais são detetadas. O B- SIPS correlaciona deteção de anomalias *host-based* com o Snort IDS que fornece deteção baseada em assinaturas de ataque.

13. Em 2009, Schmidt et al. [51] monitorizaram um *smartphone* com sistema operativo *Symbian*, extraindo características que descrevem o estado do dispositivo e que podem ser usadas para a deteção de anomalias. Esses recursos foram recolhidos por um cliente de monitorização *Symbian* e enviados ao *Remote Anomaly Detection System* (RADS). Os dados recolhidos foram analisados com a finalidade de distinguir entre o comportamento normal e anormal, tendo indicado os resultados que a maioria dos aplicativos preferidos pelos

utilizadores de telemóveis, afetavam os recursos monitorizados de diferentes maneiras.

14. Também em 2009, Hwang et al. [52] avaliaram a eficácia de autenticação baseada em dinâmica de digitação (KDA) em dispositivos móveis. A sua avaliação empírica focada em números PIN de quatro dígitos e o método proposto resultaram numa taxa de classificação incorreta de 4%.

15. Em 2011, Damopoulos et al. [53] utilizaram um grande *data log* de dados de utilizadores com quatro algoritmos de aprendizagem de máquina diferentes, ou seja, Bayesian Networks, Função Base Radial, *K Nearest Neighbors* e *Random Forest*, para detetar o uso ilegal de um *smartphone*. Eles classificaram o comportamento dos utilizadores através de chamadas telefónicas, SMS e histórico de navegação na Web. Para preservar o anonimato dos utilizadores, cada registo foi encriptado com *hash* sobre *SHA-1*.

16. Em 2013 Hamandi et al. [54] examinaram alguns dos veredictos de projeto de mensagens que causam um conjunto de vulnerabilidades no sistema operativo *Android* e mostraram como as aplicações podem ser criados para a deteção de *malware* para evitar o abuso por essa vulnerabilidade. Essas aplicações aparecem como uma aplicação normal de mensagens e são usadas para enviar/receber mensagens curtas. Uma vez que a maioria dos operadores oferecem um serviço que permite aos utilizadores transferir crédito via SMS, o mau uso deste serviço permite transferir crédito ilegalmente. A aplicação esconde a confirmação maliciosa dos operadores de telecomunicações que podem surgir após a transação para transferência de crédito.

17. Em 2105 Colp et al. [55] discutem que *Smartphones* e *tablets* são facilmente roubados ou perdidos e isso os torna vulneráveis a ataques de memória de baixo grau, como o ataque *coldboot* utilizando um barramento (*bus*), para manter uma monitorização de memória e ataques DMA (*Direct Memory Access*). O artigo descreve ainda o *Sentry*, um sistema que permite que aplicações e módulos de sistema operativo armazenem o seu código e dados no Sistema em Chip (SoC) em vez de DRAM. Eles propõem o uso de um mecanismo que foi especialmente concebido para sistemas embutidos, mas que ainda está em uso em telefones móveis existentes, para defender aplicações e Sistemas Operativos em contradição com um subsistema de memória.

Dado o foco desta dissertação nas conexões USB e respetivas vulnerabilidades, seguidamente são apresentadas as duas principais vertentes de ataques neste âmbito: (1) Ataques USB baseados em ADB e (2) Ataques baseados em Dispositivos USB.

3.2 Ataques USB baseados em ADB

Como já foi dito, as conexões USB têm sido tradicionalmente confiáveis principalmente devido à proximidade física que implicam, mas também devido à presunção de que ambos os dispositivos pertenciam ao mesmo proprietário. O trabalho de Wang e Stavrou [56] foi possivelmente o primeiro a demonstrar que esta confiança poderia ser abusada. Em particular, são discutidos ataques onde um *smartphone* atua como dispositivo de interface humana e envia *keystrokes* para controlar o *host* vítima e mostra como inicializar um *smartphone* para assumir outro telefone usando um cabo especialmente criado. O mesmo artigo, também propõe mecanismos de defesa para contrariar os ataques USB. Mais tarde [57] propostas adicionais são apresentadas e discutidas.

O trabalho de Xu [58, 59] descobriu que o recurso disponível no *Android* 4.2.2 não pode fornecer proteção suficiente quando a máquina *host* conectada ao dispositivo *Android* foi comprometida. Ele apresenta uma implementação demonstrando essa vulnerabilidade. Por isso, complementa o trabalho aqui apresentado.

A personalização dos fornecedores é uma das vantagens do *Android*, mas pode resultar numa faca de dois gumes, uma vez que pode introduzir violações de segurança. Os atacantes podem explorar diferentes formas de ataque em muitas das diferentes ROMs, já que os fornecedores adicionam *softwares*, como aplicações e processos próprios do sistema, para lidar com funcionalidades como o *USB pairing*.

Esta subsecção termina com uma breve apresentação de uma vulnerabilidade já conhecida [60] que permite explorar a conexão USB, mais precisamente a uma personalização do fornecedor que permite estender o alcance dos comandos AT (*AT-tention*), onde o sistema entende e autoriza que esses comandos sejam enviados por USB.

Os comandos AT definem uma linguagem que foi inicialmente desenvolvida para ser possível comunicar com os *Hayes Smartmodem*. Hoje em dia é o idioma padrão para comunicar com alguns tipos de modems, ver Fig. 3.2.

Esses comandos permitem efetuar o *flash* de uma partição de *boot* comprometida sem o consentimento do utilizador. Isso permite obter acesso *root*, habilitar o ADB e instalar uma aplicação de vigilância que é impossível de desinstalar sem o *re-flashing* da partição de **inicialização** do *Android*.

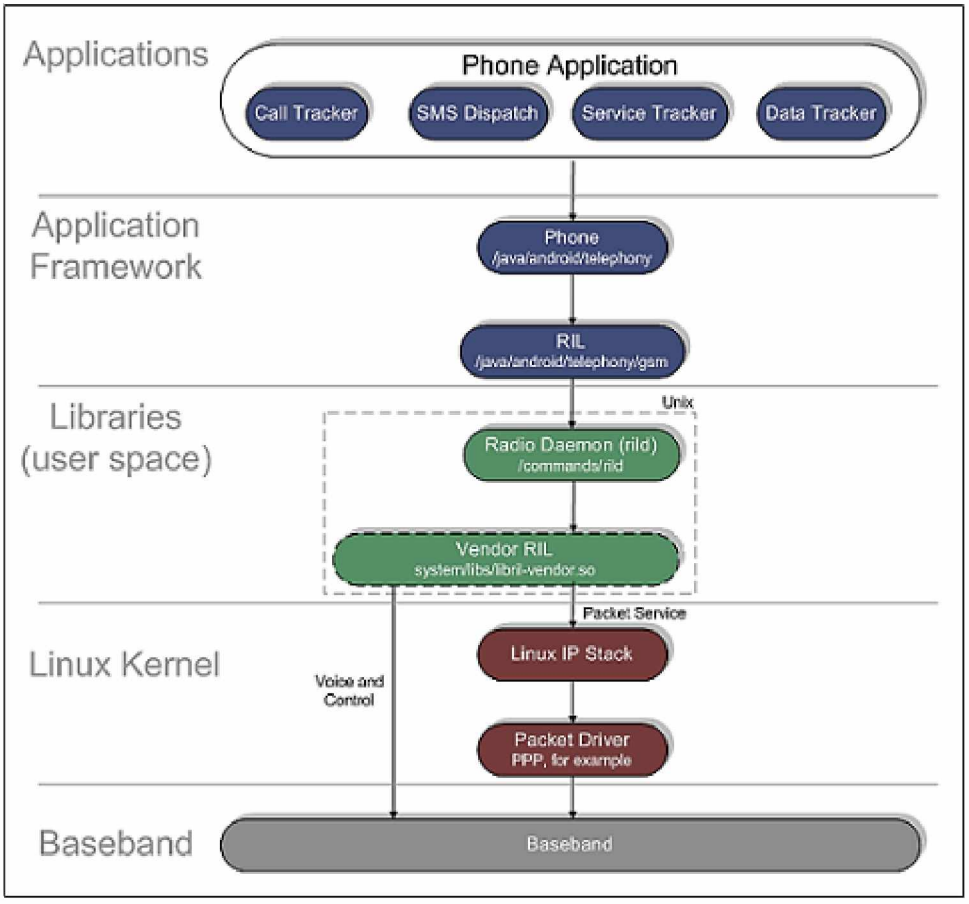

Figura 3.2: Arquitetura do sistema de telefonia *Android* (*in* [3])

No caso da **Samsung**, uma vasta lista da sua família de *smartphones* tem essa vulnerabilidade, onde foi possível comunicar com o modem através do canal USB, sem qualquer configuração anterior no dispositivo, algo que não acontece com ADB.

A **Samsung** amplia o conjunto de comandos AT padrão que vem com os padrões 3GPP e GSM, aumentando as capacidades de interação que o seu software de computador (*Kies* [61]) tem no dispositivo. O *Kies* para Windows usa o conjunto padrão e o conjunto de comandos proprietário AT expandido, para conseguir obter os contactos, o conteúdo do cartão SD e atualizar o *firmware* do dispositivo.

Através deste processo [60], é possível forçar a disponibilização da depuração USB sem autorização prévia. Desta forma, é possível utilizar o ADB para comprometer o dispositivo como mostraremos mais adiante no Capítulo 5.

3.3 Ataques baseados em Dispositivos USB

Os ataques baseados em dispositivos USB constituem outra área de pesquisa onde o objetivo está relacionado, mas alguns como simétrica ao trabalho aqui apresentado: o objetivo é atacar uma máquina (tipicamente maior) usando um dispositivo USB, mas também podem ser usados para atacar um dispositivo móvel.

Um tipo de ataque consiste em escutar as comunicações entre um dispositivo USB e um *host*. Neugschwandtner et al descreve e implementa um desses ataques (um ataque de *sniffing* USB), onde um dispositivo USB escuta passivamente todas as comunicações do *host* para outros dispositivos, sem estar situado no caminho físico entre o *host* e o dispositivo vítima. Eles também apresentam o *UScramBle*, uma solução de criptografia leve para evitar esse tipo de ataque [62].

O *USB Rubber Ducky* é um dispositivo que se assemelha a uma unidade flash USB normal (PEN). Quando conectado a um computador, é reconhecido como um teclado, mas rapidamente introduz o seu código malicioso. É uma ferramenta normalmente utilizada e muito útil por *pentesters*. Um vídeo *online* mostra como usar o dispositivo para invadir um telefone *Android* [63]. Para defender contra este tipo de Ataque (*firmware* USB malicioso em dispositivos conectados), Tian et al propôs o *GoodUSB* [64] e o *USBFILTER* [65].

As subsecções seguintes explicam o modo de funcionamento e de configuração do *USB Rubber Ducky*. Baseiam-se na documentação disponível em [66].

3.3.1 Introdução

Em 2014, a BlackHat [67] obteve muitos relatos interessantes. Um dos mais discutidos foi um relatório sobre a vulnerabilidade fatal de dispositivos USB, que permite que as unidades flash USB normais (PEN) podem ser transformadas numa ferramenta para espalhar *Malware*. O ataque foi chamado *BadUSB*, e até surgiram piadas mais tarde na Internet referindo-se ao ataque como *USBola*, comparando-o aos vírus conhecidos.

Ideias similares para a utilização de dispositivos HID para fins maliciosos já pairavam há algum tempo. Até porque seria estranho não tentar usar os dispositivos confiáveis para o OS conectados a uma interface USB. Anteriormente já havia sido escrito um artigo sobre um tema semelhante descrevendo a técnica de usar um dispositivo especial Teensy [68] para controlar um PC executando o Windows 7. O dispositivo disfarçou-se como um flash drive USB normal.

3.3.2 Pré-Requisitos

Um USB é mesmo uma Interface universal. Basta pensar em quantos dispositivos nós conectamos dessa forma: Ratos, teclados, impressoras, *scanners*, *gamepads*, *modems*, *access points*, câmaras web, telefones, etc. Sem pensar, nós ligamos o USB na tomada e o Sistema Operativo determina automaticamente o tipo de dispositivo e carrega os drivers necessários.

3.3.3 Modo de funcionamento

Na verdade, o Sistema Operativo não sabe nada sobre o dispositivo conectado. Tem que esperar até que o dispositivo diga que tipo é. Considerando um exemplo simples, quando ligamos uma unidade flash USB a uma tomada USB, a unidade flash, ver Fig. 3.3, informa o sistema operativo do seu tipo e volume.

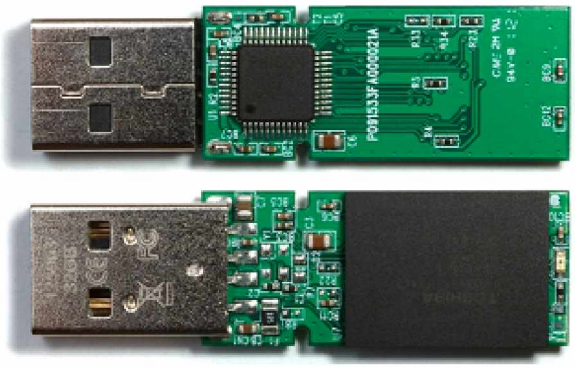

Figura 3.3: Unidade flash USB sem revestimento

3.3.4 Algoritmo de Inicialização do Dispositivo USB

A finalidade dos dispositivos USB é definida pelos códigos de classe (o USB define códigos de classe usados para identificar a funcionalidade de um dispositivo e para carregar um *driver* de dispositivo com base naquela funcionalidade. Isso permite que cada codificador de *driver* de dispositivo suporte dispositivos de diferentes fabricantes que cumprem com um código de determinada classe) comunicados ao *host* USB para a instalação dos *drivers* necessários. Os códigos de classe permitem que o *host* funcione com dispositivos de tipo único de diferentes fabricantes. O dispositivo pode suportar uma ou várias classes, cujo número é determinado pelo número de pontos de extremidade USB. Quando conectado, o *host* solicita uma gama de

detalhes padrão dos dispositivos (descritores), que usa para decidir sobre como trabalhar com o mesmo. Os descritores contêm informações sobre o fabricante e o tipo de dispositivo, que o *host* usa para selecionar o *driver* do programa.

Uma unidade flash USB normal terá o código de classe 08h, ver Fig. 3.4, (dispositivo de armazenamento em massa - MSD), enquanto uma câmara web equipada com um microfone terá dois códigos: 01h (áudio) e 0Eh (classe de dispositivo de vídeo).

	Examples	
Identifier	**USB thumb drive**	**Webcam**
Interface class	8 – Mass Storage	a. 1 – Audio b. 14 – Video
End points	0 – Control 1 – Data transfers	0 – Control 1 – Video transfers 6 – Audio transfers 7 – Video interrupts
Serial number (optional)	AA627090820000000702	0258A350

Figura 3.4: Classes do Dispositivo

Quando é conectado, o dispositivo USB é registado, recebe um endereço e envia seus descritor(es) para permitir que o Sistema Operativo instale os *drivers* necessários e envie de volta a configuração necessária. Depois disso, o *host* imediatamente começa a trabalhar com o dispositivo. Uma vez concluído o trabalho, o dispositivo é cancelado. É importante notar que os dispositivos podem ter vários descritores e também podem cancelar o seu registo e registar novamente como um dispositivo diferente. Se nós abrirmos o "corpo"de uma unidade flash USB, além do armazenamento em massa visível para o utilizador, ver Fig. 3.5, existe um controlador responsável pelas ações acima descritas.

3.3.5 O *BadUSB* e a sua Evolução Histórica

Na conferência *Black Hat*, os dois investigadores (Karsten Nohl e Jakob Lell) compartilharam a sua experiência sobre como instalar uma atualização pessoal para o *firmware* do controlador de unidade flash USB. Após algum tempo, esta unidade flash USB foi registada como um teclado e introduziu os comandos selecionados. Devido à natureza séria do problema, os investigadores decidiram não tornar o código dessa tarefa disponível. No entanto, pouco depois, outros dois investigadores (Adam Caudill e Brandon Wilson) apresentaram ao mundo inteiro na conferência Derby-

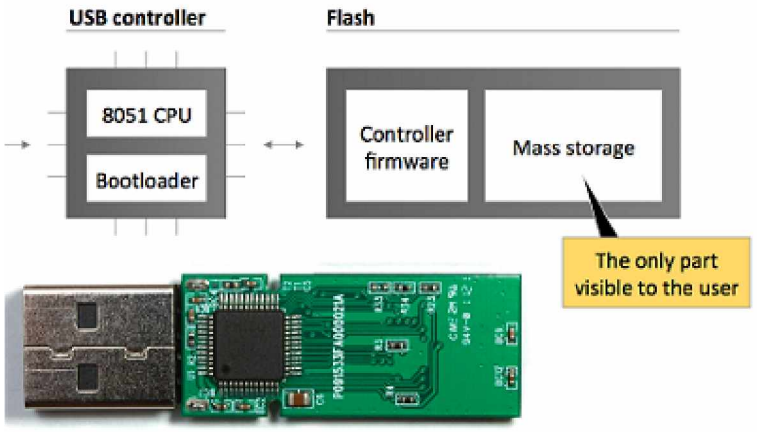

Figura 3.5: "corpo"de uma unidade *flash* USB

con [69] um PoC operável sob medida para o micro-controlador *Phison 2251-03*. O código está disponível no GitHub [70].

3.3.6 Transformação

Para transformar uma unidade flash USB numa ameaça, em primeiro lugar, é necessário um dispositivo adequado. Como o código foi enviado para um micro-controlador específico apenas, existem duas opções: ou encontrar uma unidade flash USB gerida por esse controlador, ou executar um trabalho muito desafiador de pesquisar e atualizar o *firmware* de outro micro-controlador. O exemplo que aqui será apresentado, é de um controlador bastante popular.

3.3.7 Início do Processo

Depois de encontrar o dispositivo necessário, é preciso descarregar os *sources* disponibilizados pelos investigadores:

- ***DriveCom*** — uma aplicação para comunicação com unidades flash USB Phison.
- ***EmbedPayload*** — uma aplicação para incorporar *scripts Rubber Ducky injection.bin* em *firmware* personalizado para execução subsequente quando a unidade flash USB está conectada.
- ***Injector*** — uma aplicação que extrai endereços do *firmware* e incorpora o código de correção no mesmo.

- ***Firmware*** — *firmware* personalizado 8051 escrito em Linguagem C.
- Patch — coleção de 8051 *patches* escritos em Linguagem C.

Convém referir que quando se usa *scripts Ducky*, o comando *DELAY*, que executa um atraso para um número definido de milissegundos, funcionará de forma um pouco diferente na unidade flash USB em relação à *Rubber Ducky*, e por isso será necessário ajustar o tempo de atraso.

3.3.8 Preparação do Sistema

Depois de descarregar os *sources*, como a maioria deles foram escritos em C# e exigem compilação, então vai ser necessário um compilador. Outra ferramenta que será necessária é o *Small Device C Compiler*, ou SDCC. Deve ser Instalado em `C:\Programas\SDCC`. Será necessário para compilar *firmware* e *patches*. Depois de ter compilado todas as ferramentas, é necessário verificar novamente se esta unidade flash USB é adequada para uma atualização de *firmware*:

`DriveCom.exe/drive=F/action=GetInfo`, onde '**F**' será a letra da unidade.

3.3.9 Obter a imagem do *Burner*

O próximo passo importante é selecionar uma imagem de *Burner* [71] apropriada (ficheiro binário 8051, responsável por despejar atividades e fazer o upload do *firmware* para o dispositivo). Normalmente são denominados assim:

BNxxVyyyz.BIN, onde '**xx**' é a versão do controlador (por exemplo, para PS2251-03 será 03), '**yyy**' é o número da versão (não importante), e '**z**' reflete o tamanho da página de memória e pode assumir os seguintes valores::

- **2KM** — para *2K NAND chips.*
- **4KM** — para *4K NAND chips.*
- **M** — para *8K NAND chips.*

3.3.10 Efetuar o *Dump* do *Firmware* original

Antes de iniciar este procedimento e devido à probabilidade de algo poder destruir a unidade flash USB, é altamente recomendável efetuar o *Dump* do *Firmware* original, assim se algo der errado, existe a possibilidade de tentar recuperar o dispositivo. Em primeiro lugar, deve-se mudar o dispositivo para modo de inicialização:

`tools\DriveCom.exe/drive=F/action=SetBootMode`

De seguida, utilizar o DriveCom, passando a letra da unidade, o caminho para a imagem do *Burner* e o caminho para o ficheiro onde o *dump* do *firmware* original será gravado:

`tools\DriveCom.exe/drive=F/action=DumpFirmware/burner=BN03V104M.BIN/firmware=fw.bin`

Se tudo tiver sido feito corretamente, o *firmware* de origem será gravado no ficheiro fw.bin

Para verificar qual o controlador que está instalado na unidade flash USB, pode-se usar o utilitário *usbflashinfo* [72].

3.3.11 Preparação do *Payload*

Neste momento, com o dispositivo salvaguardado, podemos pensar sobre as funções que queremos ter na nossa unidade flash USB. A Teensy tem um *kit* de ferramentas Kautilya (Ferramenta para fácil utilização de dispositivos de interface humana para testes de segurança e penetração ofensiva), que pode ser utilizado para criar *Payloads* automaticamente. Para *USB Rubber Ducky*, há um *site online* [73], com uma *interface* amigável, que permite criar *scripts* para o dispositivo, complementando a informação já existente, na lista de *scripts* concluídos, que estão disponíveis no projeto do GitHub [74].

Felizmente, os *Ducky scripts* podem ser convertidos em binário para serem incorporados em *firmware*. Para fazer isso, é necessário um utilitário *Duck Encoder* [75]. Quanto aos *scripts*, existem várias opções:

- Pode-se criar o próprio o *script*, uma vez que a sintaxe utilizada é relativamente fácil de dominar (ver o site oficial do projeto);
- Utilizar um da lista de *scripts* concluídos que estão no GitHub. Como por exemplo criar uma *reverse shell*, onde provavelmente só será necessário efetuar pequenas correções e converte-las em forma binária;
- Ou usar o *site* acima mencionado, que permite avançar passo a passo através de todas as configurações e permitirá descarregar o *script* na forma de um *Ducky script* (ou já em formato binário convertido).

Para converter o *script* em binário, é necessário executar o seguinte comando:

java -jar duckencoder.java -i keys.txt -o inject.bin, onde '**keys.txt**' é o *Ducky script* e '**inject.bin**' é o código do ficheiro binário.

3.3.12 Efetuar o *Flash* do *Firmware*

Assim que é obtido o *Payload*, devemos incorporá-lo no *firmware*. Isso é feito com os seguintes comandos:

copy CFW.bin hid.bin

tools\EmbedPayload.exe inject.bin hid.bin

O *firmware* é primeiro copiado para o **hid.bin**, e só depois é que é efetuado o *flash*. Isso ocorre porque o *Payload* só pode ser incorporado no *firmware* uma vez, portanto, o original CFW.bin deve ser deixado intocável. Após esta manipulação, obtemos um ficheiro **hid.bin** de *firmware* personalizado, com um *Payload* incorporado e basta apenas colocá-lo na unidade flash USB:

tools\DriveCom.exe /drive=F /action=SendFirmware

/burner=BN03V104M.BIN /firmware=hid.bin, onde '**F**' será a letra da unidade.

3.3.13 Opções Alternativas

Para além de usar a natureza HID da unidade flash USB para transformá-la num "teclado"e injetar os nossos *Payloads*, existem algumas outras funcionalidades que podem ser executadas. Por exemplo, é possível criar uma partição oculta no dispositivo, diminuindo o espaço visto pelo sistema operativo. Para isso, é necessário primeiro, determinar o número de blocos lógicos no dispositivo:

tools\DriveCom.exe /drive=E /action=GetNumLBAs

De seguida, localizar o ficheiro **base.c** na pasta de *patches*, retirar o comentário da linha **#define FEATURE_EXPOSE_HIDDEN_PARTITION** e adicionar outro comando define, que define um novo número LBA: **#define NUM_LBAS 0xE6C980UL** (este número deve ser par, ou então se por acaso, estiver algo como, 0xE6C981 na etapa anterior, pode-se diminuir o número para 0xE6C940, por exemplo). Depois de ter editado o código, é necessário colocar o *firmware* na pasta de *patches*, dar o nome de **fw.bin** e executar o **build.bat**, que criará um ficheiro **fw.bin** modificado no patch\bin\. Agora pode-se efetuar o *flash* na unidade flash USB. As opções **Password Patch** e **No Boot Mode Patch** são feitas da mesma maneira.

3.3.14 Recuperação do dispositivo

Se a experiência ocorrer de forma errada e a unidade flash USB se comportar de forma estranha, é possível tentar trazê-la de volta para a vida, trocando-a manualmente no modo de inicialização e usando o utilitário para restaurar o *firmware*

original, ver Fig. 3.6. Para fazer isso, antes de a conectar, é necessário fechar os contactos 1 e 2 (às vezes 2 e 3) do controlador, que estão localizados diagonalmente a partir do ponto (ver imagem).

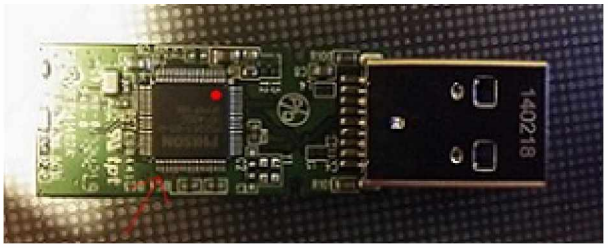

Figura 3.6: Mudar a unidade flash USB para o modo de inicialização fechando os contactos mostrados

Depois pode-se tentar trazer o dispositivo de volta à vida usando o utilitário Phison Mpall [76].

No seguinte capítulo serão apresentados os principais mecanismos de segurança e os principais melhoramentos de segurança de cada versão do *Android*.

Capítulo 4

Mecanismos de Segurança no *Android*

Neste capítulo serão apresentados os principais mecanismos de segurança existentes no Sistema Operativo *Android*.

Ao considerar a aplicabilidade de uma medida de segurança, deve-se determinar quem irá implementá-la e qual deve ser a abordagem para a sua realização. Tendo em conta estes dois fatores, é possível agrupar as medidas de segurança *Android* em três tipos, em termos da abordagem para a sua realização:

Modificações no Sistema - Requer alterações ao código-fonte do núcleo do *Android*, incluindo o *Framework*, o *Dalvik VM*, o *Linux Kernel*, os *daemons*, as bibliotecas nativas, etc. A vantagem principal é a capacidade de adicionar novas funcionalidades ao sistema e alterar o seu comportamento existente. A desvantagem principal é que se torna relativamente caro em termos de recursos humanos e de tempo, exigindo rigorosos testes e validação.

***Add-on's* do Sistema** - Requer modificações de ficheiros de configuração do núcleo do *Android*, alterações de aplicações que requerem níveis de permissão de *Signature* ou *SignatureOrSystem* e substituição de aplicações na imagem do sistema.

O Sistema de abordagem de *Add-on's*, fornece privilégios mais elevados, quer através de permissões adicionais (Assinatura e Sistema) ou executando como *root* o acesso completo ao sistema de ficheiros, aplicando a política de *firewall*, gerindo a cota de memória, ou "escutando"as teclas digitadas no ecrã táctil.

***Add-on's* de Aplicações** - Pode ser aplicado por qualquer utilizador, basta instalar uma aplicação. A aplicação *Add-on* está no mesmo nível de status que

as outras aplicações e consequentemente, pode solicitar exatamente as mesmas permissões. Os *Add-on's* de aplicações de segurança permitem uma portabilidade simples para qualquer dispositivo *Android* e facilitam a simplicidade de manutenção, embora a maioria das funcionalidades necessárias para a aplicação de segurança não estejam disponíveis e possuem capacidades limitadas de política de execução, como por exemplo o utilizador poder remover a aplicação.

4.1 Melhorias de Segurança

Desde que começaram a ser detetadas as primeiras vulnerabilidades em Sistemas *Android*, a *Google* [77] iniciou um processo de melhoria continua das suas capacidades de segurança. Segue-se um resumo das funcionalidades adicionadas em diversas versões do Android.

4.1.1 *Android* 1.5 a 4.1

Aperfeiçoamentos de segurança introduzidos nas versões do Android 1.5 a 4.1:

Android 1.5

- *ProPolice* para evitar excessos de *buffer* na *stack* (*-fstack-protector*).
- *Safe_iop* para reduzir *overflows* de inteiros.
- Extensões para *OpenBSD dlmalloc* para prevenir vulnerabilidades *double free()* e para evitar ataques de consolidação de *chunk*. Os ataques de consolidação do *Chunk* são uma maneira comum de explorar a corrupção da *heap*.
- *Calloc* do *OpenBSD* para evitar *overflows* de inteiros durante a alocação de memória.

Android 2.3

- Proteções de vulnerabilidade de formato de *String* (*-Wformat-security -Werror = format-security*).
- *Hardware-based No eXecute* (NX) para impedir a execução de código na *stack* e *heap*.

- *Linux mmap_min_addr* para atenuar o escalonamento de privilégios de "desreferência"de apontador nulo (aprimorado ainda mais no Android 4.1).

***Android* 4.0**

- *Address Space Layout Randomization* (ASLR) para definir aleatoriamente locais-chave na memória.

***Android* 4.1**

- Suporte (PIE) *Position Independent Executable.*
- Relocalizações apenas de leitura/vinculação imediata (*-Wl,-z,relro -Wl,-z,now*).
- *dmesg_restrict enabled* (Evitar perdas/fugas de endereços do *kernel*).
- *kptr_restrict enabled* (Evitar perdas/fugas de endereços do *kernel*).

4.1.2 *Android* 4.2

- **Verificação da aplicação** - Os utilizadores podem optar por ativar a opção "Verificar aplicações"e fazer com que as mesmas sejam analisadas por um verificador de aplicações antes da instalação. A verificação da aplicação pode alertar o utilizador se este tentar instalar uma aplicação que pode ser prejudicial, ou se uma aplicação for potencialmente prejudicial, poderá bloquear a sua instalação.
- **Mais controle de SMS *premium*** - O *Android* fornecerá uma notificação se uma aplicação tentar enviar SMS para um código curto que use serviços *premium* que podem causar cobranças adicionais. O utilizador pode escolher entre permitir que o aplicação envie a mensagem ou bloqueá-la.
- **VPN sempre ligado** - A VPN pode ser configurada para que as aplicações não tenham acesso à rede até que uma conexão VPN seja estabelecida. Isso impede que estas enviem dados noutras redes.
- **Certificação Fixa/*pinning*** - As bibliotecas principais do *Android* agora suportam a fixação de certificados. Os domínios marcados receberão uma falha de validação de certificado se o certificado não pertencer a um conjunto de certificados esperados. Isso protege contra possíveis compromissos das autoridades de certificação.

- **Melhor exibição das permissões do *Android*** - As permissões foram organizadas em grupos para serem mais facilmente compreendidos pelos utilizadores. Durante a revisão das permissões, o utilizador pode clicar na permissão para ver informações mais detalhadas sobre a mesma.

- ***installd hardening*** - O *daemon installd* não é executado como utilizador *root*, reduzindo a superfície de ataque potencial para a ordenação de privilégios de *root*.

- ***init script hardening*** - Os *scripts* de inicialização agora aplicam a semântica *O_NOFOLLOW* para evitar ataques relacionados com o *simlink*.

- ***FORTIFY_SOURCE*** - O *Android* agora implementa o *FORTIFY_SOURCE*. É usado por bibliotecas de sistema e aplicações para evitar corrupção de memória.

- **Configuração padrão do *ContentProvider*** - As aplicações que têm como alvo o nível 17 da API terão *export* definida como *false* por padrão para cada provedor de conteúdo, reduzindo a superfície de ataque padrão para aplicações.

- **Criptografia** - Modificação das implementações padrão de *SecureRandom* e *Cipher.RSA* para usar *OpenSSL*. Adição do suporte *SSL Socket* para *TLSv1.1* e *TLSv1.2* utilizando *OpenSSL 1.0.1*.

- **Correções de segurança** - As bibliotecas *open-source* foram atualizadas com correções de segurança que incluem *WebKit*, *libpng*, *OpenSSL* e *LibXML*. O *Android* 4.2 também inclui correções para vulnerabilidades específicas do mesmo. Informações sobre essas vulnerabilidades foram fornecidas aos membros da *Open Handset Alliance* e as correções estão disponíveis no *Android Open Source Project*. Para melhorar a segurança, alguns dispositivos com versões anteriores também podem incluir essas correções.

4.1.3 *Android* 4.3

- ***Android sandbox* reforçada com *SELinux*** - Esta versão fortalece a *Android sandbox* usando o sistema de controle de acesso obrigatório *SELinux* (MAC) no *kernel* do Linux. O reforço com *SELinux* é invisível para os utilizadores e programadores e adiciona robustez ao modelo de segurança *Android* existente, mantendo a compatibilidade com as aplicações existentes. Para garantir compatibilidade contínua esta versão permite o uso do *SELinux* num

modo permissivo. Este modo regista quaisquer violações de política, mas não interromperá as aplicações nem afetará o comportamento do sistema.

- **Nenhum programa *setuid/setgid*** - Adição de suporte para recursos de ficheiros de sistema *Android* e remoção de todos os programas *setuid/setgid*. Isso reduz a superfície de ataques *root* e a probabilidade de vulnerabilidades de segurança.

- **Autenticação ADB** - A partir do *Android* 4.2.2, as conexões com o ADB são autenticadas com um par de chaves RSA. Isso evita o uso não autorizado do ADB, onde o invasor tem acesso físico a um dispositivo.

- ***Setuid* restrito para aplicações *Android*** - A partição de sistema está agora montada como *nosuid* para processos *zygote-spawned*, impedindo as aplicações Android de executar programas *setuid*. Isso reduz a superfície de ataques *root* e a probabilidade de vulnerabilidades de segurança.

- **Limite de Capacidade** - O *Android zygote* e o ADB agora usam *prctl (PR_CAPBSET_DROP)* para descartar recursos desnecessários antes de executar as aplicações. Isso evita que sejam lançados a partir da shell e adquiram recursos privilegiados.

- **Fornecedor *AndroidKeyStore*** - O *Android* agora tem um provedor de *keystore* que permite que as aplicações criem chaves de uso exclusivo. Isso fornece aplicações com uma API para criar ou armazenar chaves particulares que não podem ser usadas por outras.

- ***KeyChain isBoundKeyAlgorithm*** - A API de *Keychain* agora fornece um método (*isBoundKeyType*) que permite que as aplicações confirmem que chaves de todo o sistema estão vinculadas a uma raiz de *hardware* de confiança para o dispositivo. Isso fornece um local para criar ou armazenar chaves particulares que não podem ser exportadas fora do mesmo, até no caso de um comprometimento de *root*.

- ***NO_NEW_PRIVS*** - O *Android zygote* agora usa *prctl* (*PR_SET_NO_NEW_PRIVS*) para bloquear a adição de novos privilégios antes do código de execução da aplicação. Isso evita que aplicações do *Android* executem operações que podem elevar privilégios por meio do *execve*. (Isso requer o *Linux kernel* versão 3.5 ou superior).

- **Aprimoramentos FORTIFY_SOURCE** - Ativação do *FORTIFY_SOURCE* no *Android x86* e *MIPS* e fortificação das chamadas de *strchr()*, *strrchr()*, *strlen()* e *umask()*. Isso pode detetar potenciais vulnerabilidades de corrupção de memória ou constantes de *string* não-terminadas.

- **Proteções de re-alocação** - Ativação apenas das re-alocações de leitura (*relro*) para executáveis vinculados estaticamente e remoção de todas as deslocalizações de texto no código do *Android*. Isso fornece defesa em profundidade contra potenciais vulnerabilidades de corrupção de memória.

- ***EntropyMixer* melhorado** - O *EntropyMixer* agora grava entropia no *shutdown/reboot*, além de mistura periódica. Isso permite a retenção de toda a entropia gerada enquanto os dispositivos são ligados e é especialmente útil para dispositivos que são reinicializados imediatamente após o aprovisionamento.

- **Correções de segurança** - O *Android* 4.3 também inclui correções para vulnerabilidades específicas do mesmo. Informações sobre essas vulnerabilidades foram fornecidas aos membros da *Open Handset Alliance* e as correções estão disponíveis no *Android Open Source Project*. Para melhorar a segurança, alguns dispositivos com versões anteriores também podem incluir essas correções.

4.1.4 *Android* 4.4

- ***Android sandbox* reforçada com *SELinux*** - O *Android* a partir desta versão usa o *SELinux* no modo de imposição. O *SELinux* é um sistema de controle de acesso obrigatório (MAC) no *kernel Linux* usado para aumentar o modelo de segurança de controle de acesso discricionário (DAC) existente. Isso fornece proteção adicional contra potenciais vulnerabilidades de segurança.

- **VPN por utilizador** - Nos dispositivos multi-utilizador, as VPNs são agora aplicadas por utilizador. Isso pode permitir que um utilizador encaminhe todo o tráfego de rede através de uma VPN sem afetar outros utilizadores no dispositivo.

- **Suporte de *ECDSA Provider* no *AndroidKeyStore*** - O *Android* agora tem um provedor de *keystore* que permite o uso de algoritmos ECDSA e DSA.

- **Avisos de monitorização de dispositivos** - O *Android* fornece aos utilizadores um aviso, se algum certificado tiver sido adicionado ao armazenamento de certificados do dispositivo, que possa permitir a monitorização do tráfego de rede encriptado.

- ***FORTIFY_SOURCE*** - O *Android* agora suporta *FORTIFY_SOURCE* de nível 2 e todo o código é compilado com essas proteções. O *FORTIFY_SOURCE* foi aprimorado para trabalhar com *clang*.

- **Certificação Fixa/*pinning*** - O *Android* 4.4 deteta e impede o uso de certificados fraudulentos do *Google* usados em comunicações SSL/TLS seguras.

- **Correções de segurança** - O *Android* 4.4 também inclui correções para vulnerabilidades específicas do mesmo. Informações sobre essas vulnerabilidades foram fornecidas aos membros da *Open Handset Alliance* e as correções estão disponíveis no *Android Open Source Project*. Para melhorar a segurança, alguns dispositivos com versões anteriores também podem incluir essas correções.

4.1.5 *Android* 5.0

- **Encriptação por defeito/padrão** - Em dispositivos que são fornecidos com *L out-of-the-box*, a encriptação completa do disco é habilitada por padrão para melhorar a proteção de dados em dispositivos perdidos ou roubados. Os dispositivos que se atualizam para *L* podem ser encriptados em **Configurações → Segurança**

- **Encriptação completa do disco melhorada** - A senha do utilizador é protegida contra ataques de força bruta usando o *scrypt* e, quando disponível, a chave está vinculada ao armazenamento de chaves de *hardware* para evitar ataques fora do dispositivo. Como sempre, o segredo de bloqueio de tela do *Android* e a chave de encriptação do dispositivo não são enviados para fora do dispositivo ou expostos a qualquer aplicação.

- ***Android sandbox* reforçada com *SELinux*** - O *Android* agora requer SELinux no modo de imposição para todos os domínios. O *SELinux* é um sistema de controle de acesso obrigatório (MAC) no *kernel Linux* usado para aumentar o modelo de segurança de controle de acesso discricionário (DAC) existente. Esta nova camada fornece proteção adicional contra potenciais vulnerabilidades de segurança.

- ***Smart Lock*** - O *Android* agora inclui *trustlets* que oferecem mais flexibilidade para desbloquear dispositivos. Por exemplo, os *trustlets* podem permitir que os dispositivos sejam desbloqueados automaticamente quando estão perto de outro dispositivo confiável (via NFC, *Bluetooth*) ou sendo usados por alguém com uma "face"confiável.

- **Multi-utilizador, perfil restrito e modos de convidado para telefones e *tablets*** - O *Android* agora oferece vários utilizadores no telefone e inclui um modo de convidado que pode ser usado para fornecer acesso temporário fácil ao dispositivo sem conceder acesso aos seus dados e aplicações.

- **Atualizações para o *WebView* sem OTA** - O *WebView* agora pode ser atualizado independente da estrutura e sem um sistema OTA. Isso permitirá uma resposta mais rápida a possíveis problemas de segurança no mesmo.

- **Encriptação atualizada para HTTPS e TLS/SSL** - O *TLSv1.2* e *TLSv1.1* agora estão ativados e o *Forward Secrecy* é preferencialmente escolhido, o *AES-GCM* está também ativado e os conjuntos de codificação (*MD5*, *3DES* e conjuntos de cifra de exportação) estão desativados [78].

- **Suporte *non-PIE linker* removido** - O *Android* agora requer todos os executáveis vinculados dinamicamente para suportar PIE(executáveis independentes de posição). Isso melhora a implementação de aleatoriedade de *layout* de espaço de endereço do *Android* (ASLR).

- **Melhorias no *FORTIFY_SOURCE*** - As seguintes funções de *libc* agora implementam as proteções *FORTIFY_SOURCE*: *stpcpy()*, *stpncpy()*, *read()*, *recvfrom()*, *FD_CLR()*, *FD_SET()* e *FD_ISSET()*. Isso fornece proteção contra vulnerabilidades de corrupção de memória envolvendo essas funções.

- **Correções de segurança** - O *Android* 5.0 também inclui correções para vulnerabilidades específicas do mesmo. Informações sobre essas vulnerabilidades foram fornecidas aos membros da *Open Handset Alliance* e as correções estão disponíveis no *Android Open Source Project*. Para melhorar a segurança, alguns dispositivos com versões anteriores também podem incluir essas correções.

4.1.6 *Android* 6.0

- **Permissões de tempo de execução** - As solicitações de permissões das aplicações são concedidas no tempo de execução em vez de serem concedidas no momento da instalação da mesma. Os utilizadores podem ativar e desativar permissões para aplicações M e pré-M.

- **Inicialização do arranque verificada** - Um conjunto de verificações encriptadas do *software* do sistema são realizadas antes da execução para garantir que o telefone esteja saudável desde o *bootloader* até o sistema operativo.

- **Segurança do *Hardware*-isolado** - Nova *Hardware Abstraction Layer* (HAL) usada pela *Fingerprint API*, *Lockscreen*, *Device Encryption* e *Client Certificates* para proteger as chaves contra compromissos do *kernel* e/ou ataques físicos locais.

- **Impressões digitais** - Os dispositivos podem agora ser desbloqueados com apenas um toque. Os programadores também podem aproveitar as novas APIs para usar impressões digitais para bloquear e desbloquear chaves de encriptação.

- **Opção de cartão SD** - O cartão removível pode ser selecionado por um dispositivo e expandir o armazenamento disponível para dados locais de *apps*, fotos, vídeos, etc., mas ainda ser protegido por encriptação de *block-level*.

- **Limpar tráfego de texto** - Os programadores podem usar um novo *StrictMode* para certificar-se de que a sua aplicação não usa texto não encriptado.

- ***System Hardening*** - Endurecimento do sistema através de políticas aplicadas pelo *SELinux*. Isso oferece um melhor isolamento entre os utilizadores, filtragem *IOCTL*, reduzindo a ameaça de serviços expostos, apertando ainda mais o acesso aos domínios *SELinux* e ao *limited/proc*.

- **Controle de Acesso USB** - Os utilizadores devem confirmar antes de permitir o acesso USB a ficheiros, armazenamento ou outras funcionalidades no telefone. O padrão agora é permitir apenas o acesso ao armazenamento que requer aprovação explícita do utilizador.

4.1.7 *Android* 7.0

- **Encriptação *File-Based*** - A encriptação no nível do ficheiro, em vez de encriptar toda a área de armazenamento como uma única unidade, isola e protege melhor os utilizadores e perfis individuais (como pessoais e de trabalho) num dispositivo.

- **Inicialização do arranque direta** - Ativado por encriptação *File-Based*, o *Direct Boot* permite que determinadas aplicações, como o despertador e recursos de acessibilidade, sejam executados quando o dispositivo for ligado, mas não desbloqueado.

- **Verificação da inicialização do arranque** - A inicialização verificada agora é estritamente imposta para impedir que dispositivos comprometidos sejam

inicializados. O dispositivo suporta correção de erros para melhorar a confiabilidade contra corrupção de dados que não sejam mal-intencionados.

- ***SELinux*** - A configuração atualizada do *SELinux* e a maior cobertura de *seccomp* bloqueiam a *sandbox* da aplicação e reduzem a superfície de ataque.
- **Aleatoriedade da biblioteca *load-Order* e *ASLR* melhorado** - Maior aleatoriedade torna alguns ataques de reutilização de código menos confiáveis.
- ***Kernel hardening*** - Adicionada proteção de memória adicional para *Kernels* mais recentes, marcando porções de memória do *Kernel* como apenas de leitura, restringindo o acesso do mesmo a endereços de espaço de utilizador e reduzindo ainda mais a superfície de ataque existente.
- **Esquema de assinatura *APK v2*** - Foi introduzido um esquema de assinatura para todos os ficheiros que melhora a velocidade de verificação e reforça as garantias de integridade.
- **Loja de CA's confiáveis** - Para tornar mais fácil que as aplicações controlem o acesso ao seu tráfego de rede seguro, as autoridades de certificação instaladas pelo utilizador e as instaladas por meio das *APIs* de administração de dispositivos deixam de ser confiáveis por defeito no segmento para o *API Level 24+*. Além disso, todos os novos certificados de dispositivos *Android* devem ser fornecidos pela mesma loja de CA's confiáveis.
- **Configuração de segurança de rede** - Configurar a segurança da rede e TLS através de um ficheiro de configuração declarada.

4.2 *Anti-Malware*

Para identificar e remover *malware*, o software *anti-malware* para dispositivos móveis examina todos os ficheiros em locais especificados, anexos de e-mail, memória, configuração do sistema, *Multimedia Messaging Service* (MMS), objetos Bluetooth e outras áreas relevantes. Geralmente identifica e extermina *malware* com conhecimento baseado num repositório de assinaturas.

Como descrito anteriormente, várias soluções comerciais estão disponíveis para o *Android*, que também fornece um componente *anti-malware*. Existem também antivírus *open-source* e detetores de *rootkit* como por exemplo o **ClamAV** [79]. O **ClamAV** está disponível para sistemas operativos semelhantes a Linux sob os termos da GNU GPL e as assinaturas estão disponíveis gratuitamente. Geralmente é

executado como um *daemon* e atende a solicitações de inspeção por outros processos, é útil principalmente como um scanner de anexos de e-mail e um servidor de ficheiros de rede/proxy [51].

O *Anti-Malware* é uma solução bem conhecida e é amplamente utilizada noutras plataformas. As soluções baseadas em assinaturas podem fornecer falsos negativos, mas apenas detetam *malware* conhecido e exigem a atualização contínua do repositório de assinaturas. Neste momento, a solução *anti-malware* poderá não ser 100% eficaz para dispositivos móveis.

4.3 *Firewalls*

Uma *firewall* a correr em dispositivos comandados pelo *Android* pode limitar a vulnerabilidade a ataques remotos, impedindo o *scanning* com base na Internet e o acesso a serviços internos. A solução **SMobile** para *Android* também alega incluir uma ferramenta de *firewall*. O Linux 2.6 inclui a *firewall* integral do *iptables* que usa o *framework NetFilter*. *NetFilter* é um subsistema de *kernel Linux* que fornece filtragem de pacotes, reescrita e capacidades de rastreamento de conexão que são usadas para implementar *firewalls*. As capacidades do *iptables/NetFilter* e dos módulos associados incluem: filtragem de pacotes de entrada, saída ou encaminhamento; Correspondência de pacotes por correspondência de regras em campos de protocolo (por exemplo, protocolo IP, endereço src/dest); inspeção *stateful*; i descarta, rejeita ou aceita um pacote com uma notificação ICMP.

O *NetFilter* está habilitado no T-Mobile G1, portanto é necessário apenas uma aplicação de controlo. Devido à exigência de permissões para atualizar a política da *firewall*, a aplicação de controlo pode ser implementada como um complemento do sistema e depende da aplicação "su"do *firmware*. "Su"é um comando que permite executar como "root"e neste caso é usado para executar o comando *iptables* com os privilégios de super-utilizador necessários.

Enquanto uma *firewall* é uma solução bem conhecida e altamente eficaz que pode ser amplamente utilizada noutras plataformas, não protegerá contra ataques via *browser*, SMS/MMS, e-mail ou Bluetooth e não fornecerá filtragem de chamadas telefónicas.

A *firewall* básica que é ativada nas regras do *Android* é muito simples e fornece a capacidade de bloquear/negar comunicação a endereços IP, DNS e portas específicas. A política de *firewall* mais adequada para o *Android* deve ser semelhante à *firewall* do SO Windows, que permite definir regras no nível da aplicação. De tal forma é possível definir para cada aplicação quem pode acede-lo e qual a aplicação que

pode enviar informações e para onde. Podemos ter certeza de que o *scanning* de portas não é pré-definido a partir do dispositivo por uma aplicação mal-intencionada, bloqueando o acesso a sites específicos e muito mais.

4.4 Sistema de deteção e prevenção de intrusões (*IDS/IPS-Intrusion Detection/Prevention System*

Os recursos de IDS/IPS adaptados para *smartphones* foram oferecidos nos últimos anos pelos principais fornecedores de segurança. Como um exemplo, a *Trend Micro Mobile Security* [80] afirma que os seus produtos defendem contra *malware* e ajudam a evitar invasões e fugas de dados indesejados através de novas tecnologias de deteção de *firewall* e intrusão. O *Norton Smartphone Security* pela *Symantec* fornece recursos IDS para o *Symbian* e o *Windows-Mobile* [81].

O *Linux Intrusion Detection System* (LIDS) é um aprimoramento para o *kernel* do Linux (implementado como um *patch* do *kernel*). Implementa vários recursos de segurança, como o *Mandatory Access Control* (MAC), um detetor de *scan* de portas, proteção de ficheiros e proteção de processos (restringindo até o "root"). O *OSSEC HIDS* é um sistema de deteção de intrusões *open-source* para Linux que executa análises de log, verificação de integridade de ficheiros, deteção de *root-kit*, alertas em tempo real e resposta ativa.

4.5 Controlo de Acesso

O *Android* incorpora vários mecanismos de controlo de acesso. Enquanto esses mecanismos são administrados no nível da aplicação ou apenas em ficheiros, o Linux pode fornecer outras ferramentas que são diretamente aplicadas pelo *kernel*, como por exemplo, o *Linux Security Modules* (LSM) que permite que o *kernel* Linux suporte uma variedade de modelos de controle de acesso.

Existem dispositivos *Android* G1 com o Security-Enhanced Linux (SELinux), que é a implementação mais conhecida de um *Linux Security Module* (LSM) [28]. O *SELinux* permite a restrição de qualquer processo no sistema, incluindo o "root", limitando o acesso de processos e utilizadores a recursos e/ou serviços, limitando assim os danos potenciais causados por aplicações mal-intencionadas ou *exploits*. As suas decisões são baseadas numa política de controlo de acesso, que deve ser implementada em conjunto com o sistema base.

A questão principal que o *SELinux* procura resolver é o confinamento dos processos do sistema, particularmente aqueles com altos privilégios. Uma vez que todos os processos do sistema são conhecidos à partida, a política especificaria exatamente quais as ações legais que cada serviço pode executar. Caso uma vulnerabilidade seja encontrada num desses serviços, o *SELinux* limita o processo e, portanto, limita a manobrabilidade do invasor. O *SELinux* é considerado uma medida preventiva.

De um modo geral, todas as atividades normais no dispositivo devem ser aprovadas pela política *SELinux*, e qualquer negação durante a operação normal deve ser considerada como um*bug*. Além disso, não há interação do utilizador e, portanto, nenhum impacto na usabilidade. A integração da política *SELinux*, não requer a implementação de modificações ao *kernel* (ou seja, modificação do sistema). O *SELinux* no *Android* consome poucos recursos e tem custos muito baixos.

4.6 *Login* do proprietário

Em redes de computadores privadas e públicas (incluindo a Internet), a autenticação é normalmente feita através do uso de senhas de *login*. Um mecanismo semelhante pode ser desenvolvido para o *Android*. Ao ligar um dispositivo, o utilizador deverá inserir um código (por exemplo, uma senha ou uma forma usando o *touchscreen*) conforme especificado durante a configuração do dispositivo e conhecido apenas pelo próprio.

O dispositivo pode ser re-bloqueado após várias tentativas erradas ou, automaticamente após um período de inatividade. A fim de usá-lo novamente o proprietário será obrigado a introduzir o código de acesso. Uma característica adicional para proteger dados sensíveis seria o de bloquear o dispositivo através de um SMS especifico que o utilizador pode enviar caso o dispositivo seja roubado. Os meios biométricos para identificação (ou seja, digitalização de impressões digitais ou utilizando a câmara do *smartphone*) também estão a começar a emergir para dispositivos móveis.

Embora estas soluções baseadas em códigos possam ser irritantes para o utilizador (a necessidade de o introduzir para cada uso), elas podem não proteger todas as informações (por exemplo, os dados no cartão SD externo estão ainda expostos) e senhas/códigos podem sempre ser quebrados. No momento, o *Android* é fornecido com um mecanismo de padrão de bloqueio de ecrã simples.

4.7 Proteção de permissões *Android*

Durante a instalação de uma aplicação no *Android*, o utilizador pode ler uma lista de permissões necessárias e pode recusar a instalação com base nessa lista. Na prática, é improvável que o utilizador negue a instalação de uma aplicação que ele pretenda com base nessa lista. De seguida serão descritos dois mecanismos opcionais destinados a restringir o acesso às permissões do *Android*.

4.7.1 Permissões *Android* seletivas

Às vezes, as permissões são solicitadas apenas para recursos esotéricos da aplicação (por exemplo, um jogo pode solicitar acesso "Internet"para fazer upload de pontuações altas). Às vezes, a permissão é solicitada para um caso de uso válido, mas o utilizador não planeia utilizar esse recurso. Por exemplo, o *ChompSMS* permite enviar SMS utilizando tanto a operadora móvel como a Internet, mas um utilizador pode desejar usar apenas um dos métodos.

Se for adicionado um recurso avançado ao Instalador de Pacotes, permite ao utilizador recusar determinadas permissões solicitadas, mas ainda permitir a instalação da aplicação. Tal mudança é altamente benéfica para os utilizadores conscientes da segurança, mas não impede a usabilidade para utilizadores não conscientes. Esta solução dá proteção ao não conceder permissões desnecessárias que podem ser maliciosamente usadas. No geral, o esforço necessário é baixo, mas requer uma modificação do sistema e possíveis mudanças no projeto. Além disso, as aplicações que garantem um conjunto parcial de permissões podem falhar se o programador não antecipar e fornecer uma solução para tal situação (ou seja, lidar com casos em que as permissões parciais foram dadas) [82].

Essa solução pode ser aprimorada para aumentar as opções de segurança dos dispositivos *Android*, limitando as permissões concedidas através de uma política predefinida. O objetivo é proteger os utilizadores de conceder permissões desnecessárias que podem ser usadas maliciosamente. É principalmente relevante para utilizadores empresariais. Quando uma aplicação está prestes a ser instalada, o pacote *Installer* compararia as permissões solicitadas com a diretiva e não concederia permissões que estivessem na lista negra. A política seria definida pelo pessoal de TI da empresa que disponibilizaria o dispositivo ao empregado. Uma política sofisticada pode listar uma combinação de recursos, permitir determinadas permissões apenas para assinantes de aplicações selecionadas, limitar recursos, etc.

4.7.2 Aplicação de Gestão de Permissões

Esta solução percorre periodicamente as permissões de aplicações do *Android* e os ficheiros *world_read/write_access* para detetar instalações de aplicações desconhecidas. Ela soluciona parcialmente o problema de ID de utilizador compartilhado. A motivação para um *scanner* de permissão do *Android* é a de ajudar o utilizador a detetar aplicações com permissões indesejadas (que talvez tenham sido acidentalmente concedidas no momento da instalação).

Outro aspeto é a contenção de uma falha de segurança causada pela existência de ficheiros que podem ser acedidos e/ou modificados por todas as aplicações (permissões globais de leitura/gravação). Para solucionar tal situação, um *scanner* de permissão apresenta ao utilizador permissões concedidas a cada uma das aplicações instaladas no dispositivo. Com o auxílio de tal exibição, permissões estranhas saltariam à vista imediatamente.

Por exemplo, um reprodutor de média que tenha a permissão para ouvir eventos de teclado deve chamar a atenção do utilizador. Se uma anomalia de permissão for detetada, várias ações são possíveis: Revogar as permissões indesejadas; desinstalar a aplicação; bloquear a aplicação em execução; ou requerer a aprovação do utilizador quando a permissão é realmente usada pela aplicação, facilitando assim a aprovação de uma única execução ou de uma única sessão. Todas as ações possíveis, exceto a primeira, exigirão grandes mudanças na estrutura. Ao usar a primeira ação, o utilizador ainda teria que decidir o que fazer com a aplicação: ou usá-la ou desinstalá-la. Tal aplicação poderia ser simplesmente implementada com pouco esforço como uma aplicação adicional.

4.8 Encriptação de Dados

Um telefone móvel que é perdido ou roubado pode conter importantes dados de contactos pessoais, e-mails empresariais, ou dados confidenciais da empresa. A fim de proteger as informações sensíveis, armazenadas no dispositivo, os dados podem ser encriptados. Os cartões de memória de armazenamento amovível que estão conectados aos dispositivos também podem ser encriptados. A criptografia é baseada normalmente em senhas para aceder ao dispositivo ou dados no mesmo, os utilizadores necessitam sempre de autenticação com uma senha ou PIN. Políticas de Controlo, como limitar o número de tentativas de introdução da senha podem ser implementadas para maior segurança. Os recursos de encriptação foram incorporados a partir da versão *Cupcake* do *Android*.

4.9 Encriptação de chamadas telefónicas

Chamadas telefónicas encriptadas podem fornecer uma conexão de voz segura por meio de autenticação e encriptação e evitariam efetivamente que conversas telefónicas fossem alvo de espionagem. A encriptação de chamadas telefónicas exige que ambas as partes compartilhem a mesma aplicação e isso pode reduzir a qualidade das mesmas. São usadas principalmente para fins militares. Utilizando as permissões fornecidas pela estrutura de aplicações do *Android*, essa solução pode ser fornecida como uma aplicação complementar.

4.10 Rede Virtual Privada (VPN- *Virtual Private Network*)

Uma solução *VPN* é importante principalmente para utilizadores empresariais e fornece efetivamente uma conexão segura a redes protegidas/privadas através da Internet. Existem algumas ferramentas *VPN* para Linux, sendo o *OpenVPN* o principal concorrente *open-source*. É possível efetuar a conexão a servidores *Microsoft* através de *PPTP* (*Point-to-Point Transfer Protocol*) usando o *PPTP Client*. A *VPN* também pode ser implementada usando o *OpenSSH* (versão 4 ou posterior) sobre o protocolo *SSH*. O Linux também suporta *IPSec*, que é um método padrão para estabelecer conexões seguras que também é usado para *VPNs*. As conexões *VPN PPTP, L2TP* e *IPSec* são ativadas a partir da versão 1.6 do *Android* (Atualização da versão *Donut*). A ativação de soluções *VPN* adicionais baseadas em Linux no *Android* envolvem um esforço reduzido. Os privilégios de *root* necessários para a criação de um adaptador de rede virtual tornam essa tarefa num complemento do sistema.

4.11 Filtro de *Spam*

Um filtro de *spam* bloqueia *MMS, SMS, e-mails* indesejados e chamadas de origem não confiável. Experiência anterior desta metodologia no sector do correio eletrotécnico indica que existem dois métodos proeminentes de lidar com o *spam*. A abordagem lista branca/negra onde as fontes são conhecidas como boas (*white-list*) e, portanto, entregues, ou como más (*black-list*) e, portanto, bloqueadas. Uma segunda abordagem é fazer com que o sistema filtre os itens de entrada usando filtros de classificação de *machine learning* (ML), com base em recursos do sistema de

transporte e/ou conteúdo da mensagem. Os classificadores são treinados e atualizados ao longo do tempo.

A abordagem ML também pode usar a hora do sistema e um perfil de utilizador para uma classificação mais refinada dos itens recebidos. Na área dos telemóveis, a abordagem da lista branca/negra é mais comum, com o ID do emissor da chamada a ser usado como a fonte para permitir/bloquear uma chamada. A solução do *SMobile* que foi criada para *Android*, inclui a solução *PointGuard* que pode filtrar mensagens de *spam* e chamadas baseadas em listas negras atualizadas. A partir da versão 6.0 do *Kaspersky Mobile Anti-Virus* são fornecidos recursos *anti-spam* que bloqueiam números de telefone de fontes de *spam* conhecidas, números incorretos ou palavras ou frases indesejadas que foram adicionadas a uma lista negra. O software também suporta uma lista branca. O *Norton Smartphone Security* e o *TrendMicro Mobile Security 3.0* incluem um recurso de proteção *anti-spam* do SMS que pode bloquear mensagens curtas (SMS e MMS) de remetentes desconhecidos.

Um filtro de *spam* em dispositivos móveis requer uma atualização contínua pois pode sofrer de uma taxa de erros e falsos positivos. No entanto, tais atualizações podem consumir muitos recursos.

4.12 Certificação de Aplicações

Uma solução implementada por outros sistemas operativos móveis (ou seja, *Symbian*) e defendida no *OMTP Application Security Framework* tem níveis de confiança diferentes para as aplicações instaladas. Uma aplicação instalada no dispositivo móvel pode solicitar permissões diferentes (por exemplo, iniciar uma chamada de saída, criar uma conexão de dados de rede usando *HTTP/HTTPS*, enviar SMS ou MMS, determinar a localização atual do dispositivo usando o GPS), dependendo do "Seu"nível de confiança. O nível de confiança é atribuído à aplicação de acordo com sua origem utilizando uma assinatura e mecanismos de certificação de terceiros. Sempre que há uma tentativa de instalar uma aplicação, o primeiro passo é validar o seu certificado. Se não houver nenhum certificado ou a validação do certificado tiver falhado, a aplicação receberá o nível de confiança mais baixo e só poderá ser instalada se solicitar as permissões básicas inofensivas. Alternativamente, a instalação pode ser sempre abortada. Se o certificado for válido, as permissões solicitadas pela aplicação instalada serão concedidas ou negadas com base no nível de confiança associado. Se, por exemplo, a aplicação solicitar permissão para aceder à Internet, mas o seu nível de confiança não permitir, a permissão não será concedida e causará um erro de tempo de execução se uma API exigir que ela seja utilizada.

O *Android* usa certificados de forma limitada para garantir a integridade de um *package* e para garantir que que dois ou mais *packages* são da mesma origem. As aplicações que definem as suas próprias permissões podem optar por conceder essas permissões apenas aos *packages* assinados pelo mesmo autor. Não há suporte para *root Certificate Authorities* (CAs) ou cadeias de certificados no *Android*. Para usar o mecanismo de confiança da aplicação, o *Android* deve ser modificado para suportar níveis de confiança de aplicações, associando certificados CA ao nível de confiança e verificando cadeias de certificados. Esse mecanismo é altamente eficaz na avaliação da natureza de uma aplicação, detetando aplicações mal-intencionadas antes de serem instaladas no dispositivo e limitando qualquer dano potencial causado pelas mesmas. No entanto, esta solução é muito cara em termos de implementação e manutenção.

Embora a certificação tenha sido comprovada como muito eficaz, não é livre de erros e as aplicações mal-intencionadas podem ainda ser inadvertidamente assinadas e aprovadas. Além disso, podemos assumir que os utilizadores continuarão a descarregar e instalar aplicações "não aprovadas"que estão disponíveis em *websites* de forma gratuita em vez das aplicações confiáveis que necessitam ser pagas. Além disso, o *Android* é baseado numa abordagem *open-source* e o quadro de certificação contradiz esta abordagem. Assim, os pesquisadores devem procurar alternativas para capturar a semântica de aplicações sem depender da inspeção manual do código. Uma abordagem que está intimamente relacionada é a análise estática e a verificação do código.

4.13 Gestão de Recursos

Há quatro recursos principais num ambiente de computação, como o Android: CPU, Armazenamento, Memória RAM e Input/Output. Cada aplicação pode solicitar tanto de cada recurso quanto necessitar. Nem todos esses recursos possuem salvaguardas contra uso injusto ou indevido.

- **CPU**, o *Android* utiliza o *Completely Fair Scheduler* (CFS) do Linux que garante que uma parcela igual de CPU é distribuída entre todos os processos. Além disso, a processos específicos pode ser concedido uma quota maior, mas mesmo assim são impedidos de monopolizar a CPU. Testes acerca deste mecanismo de equidade, demonstraram que criando uma aplicação simples que inicia 100 *threads* que não fazem nada em particular e executando a mesma

num dispositivo G1 obtiveram como resultado o congelamento de todo o dispositivo.

- **Armazenamento**, o Linux suporta cotas de armazenamento, mas o *Android* não o permite. Isso significa que atualmente cada aplicação pode criar ficheiros tão grandes quanto quiser, tanto no armazenamento interno como no cartão SD [83]. A lista de aplicações mostra os tamanhos de cada uma, de modo a que seja possível desinstalar aplicações grandes, apesar de isso não ser suficiente. Os ficheiros podem ser criados fora da pasta da aplicação em particular e, portanto, não serem contabilizados nos números da lista das mesmas. Uma solução poderia passar por criar cotas de armazenamento na configuração padrão.

- **Memória RAM**, quando não há memória livre suficiente no sistema, o Linux normalmente mata um processo (quase sempre aleatoriamente) para libertar RAM. O *Android* adicionou um ainda mais agressivo *lowmemorykiller* para a mesma tarefa. Na prática, tem sido demonstrado [84] que este processo de matar pode ser invocado visitando um *website* malicioso, de tal forma que todas as aplicações no sistema serão terminadas, incluindo serviços essenciais, como o servidor do sistema. Quando o servidor do sistema é terminado, o dispositivo reinicia. Esse comportamento agressivo pode ser modificado.

- ***Input/Output***, a largura de banda da rede e de *Input/Output* do disco são limitadas. Existem vários controladores de largura de banda de *Input/Output* para Linux, mas nenhum deles está ativado no *Android*. Em relação à rede, outro exemplo desta categoria é o chamado "*traffic shaping*", mas também está desativado no *Android*. No que diz respeito a algumas aplicações que são consideradas críticas, como por exemplo a aplicação "telefone", sugerem-se ajustes para conceder fatias maiores a tais aplicações. Todos os recursos mencionados suportam definir fatias maiores para processos específicos. Todas as configurações de gestão de recursos mencionadas requerem modificações no sistema.

4.14 Gestão Remota

Os mecanismos de gestão remota consolidam vários outros mecanismos de segurança, fornecendo a capacidade de controlar remotamente, configurar e gerir o dispositivo. Isso pode incluir: configuração remota de vários parâmetros (por exemplo, confi-

gurações de rede *Wi-Fi* ou *Bluetooth*); atualizar a política de *firewall*; sugerir e forçar atualizações de segurança; atualizar ferramentas *anti-malware* e *anti-spam*; rastrear a localização do dispositivo; desinstalar/instalar aplicações; remover o dispositivo remotamente; excluir ou encriptar dados; fornecer os meios para assistência remota quando um problema é encontrado (a gestão remota é uma solução relevante, principalmente para utilizadores empresariais).

A gestão remota de dispositivos *Android* fornece uma solução centralizada para proteção de dados confidenciais em caso de perda ou roubo, deteção de intrusões e garantia de segurança atualizada. Requer suporte humano e alta manutenção e, portanto, é relativamente dispendiosa. Devido aos padrões de alta disponibilidade exigidos pelo mercado corporativo, o esforço envolvido na implementação é médio. Os privilégios necessários para o acesso à informação relevante e o controle efetivo sobre o dispositivo exigiriam uma modificação do sistema.

4.15 Segurança de *Context-Aware*

Uma solução *Context-Aware* pode permitir e restringir dinamicamente o acesso a recursos (documentos, *e-mails*) e serviços (câmara, Internet, telefone, mensagens) com base numa política predefinida e no contexto momentâneo do dispositivo. Pode proporcionar uma melhor proteção para o conteúdo confidencial e garantir a integridade de vários serviços. Aplicações com reconhecimento de contexto, como o *Locale* no *Android Market*, já estão disponíveis, mas exigem alta interação com o utilizador na definição das regras e não são orientados para a segurança. O desafio inerente a essas soluções é aprender e definir políticas automaticamente, de preferência sem forçar a interação com o utilizador. O seu esforço de implementação pode ser classificado como médio e exigiria uma modificação do sistema.

4.16 Verificação de Integridade

A verificação de integridade é usada para validar que o sistema não foi adulterado. Pode evitar a exposição de informações em cenários como substituir a aplicação do telefone por uma semelhante que contenha um *Trojan* que efetue chamadas telefónicas ou permita escutá-las. A primeira abordagem para verificação de integridade envolve a monitorização de mudanças no sistema de ficheiros em relação a um estado de base-line. O *Tripwire* [85] é um utilitário *open-source* de verificação de integridade que tipifica essa abordagem.

A segunda abordagem utiliza um chip de *hardware* e destina-se a garantir a validade do sistema a um parceiro, como a operadora telefónica ou provedores de conteúdo. Por exemplo, a IMA, a implementação da *Trusted Computing Framework* para Linux, interceta todos os ficheiros de acesso/execução e verifica sua integridade (por exemplo, o *hash* resultante de uma encriptação), utilizando um chip de *hardware* TPM (*Trusted Platform Module*) (que é protegido contra o *software* do sistema) do sistema medido. Este tipo de medição de integridade pode ser implementado no *Android* de várias formas. Este atestado permite que as empresas verifiquem remotamente que os seus telefones de funcionários não foram manipulados. Pode-se tornar a subversão do telefone mais difícil, se a inicialização segura for implementada (embora isso não esteja no espírito de abertura do *Android*). Pode permitir ao utilizador encriptar ficheiros que só estarão disponíveis num telefone específico.

4.17 Análise Estática Automática e Verificação de Código

Os ficheiros *apk* do *Android* encapsulam informações valiosas que podem ajudar na compreensão do comportamento de uma aplicação. Essas informações incluem permissões solicitadas, métodos de estrutura chamados pela mesma, classes de estrutura usadas, *widgets* de Interface de utilizador e muito mais. Utilizando o *apk* extrator de recursos, é possível extrair recursos dos ficheiros, incluindo:

1. recursos do *apk*, como o tamanho, o número de entradas zip e o número de ficheiros para cada tipo.

2. recursos XML, como o número de elementos *xml*, os atributos, os *namespaces*, as strings distintas e as permissões utilizadas no *Android Manifest*.

3. recursos *dex*, como um Booleano para cada método na estrutura (usado ou não), um Booleano para cada tipo na estrutura (o tipo é usado ou não), como por exemplo, *MotionEvent* ou *execSQL()*.

O *Android* usa um formato proprietário para o *Java bytecode* chamado *dex* [86]. Com a finalidade de extrair recursos significativos de arquivos *dex* é possível utilizar engenharia reversa no *dex file parser*, que está inserido apenas no *VM Dalvik*, e desenvolver um analisador de ficheiros "*dex*". Este analisador pode transformar o conteúdo do ficheiro *dex* em recursos padrão (por exemplo, sequências de caracteres, tipos, classes, protótipos, métodos, campos, anotações, valores estáticos, herança,

modificadores, *opcodes*) e assistir na verificação manual dos recursos das aplicações [28].

Schmidt et al. [51] descreve como monitorizar eventos no *kernel*. Isto é, identificar eventos críticos do *kernel*, do ficheiro de *log*, do sistema de ficheiros e da atividade de rede, e simultaneamente criar mecanismos eficientes para monitorizá-los num ambiente de recursos limitados. Assim ficou demonstrado o seu *framework* na análise estática de chamada de funções .

Assim, tal abordagem fica associada com a certificação e pode fornecer uma alternativa automatizada como parte do processo da mesma. Este método pode ser utilizado para o rápido exame de *packages* do *Android* e informar a equipa *Google*, através do *Android Market* sobre aplicações suspeitas.

No seguinte capítulo será apresentada a descrição do procedimento de implementação de um *script* que permite explorar vulnerabilidades.

Capítulo 5

Implementação

Neste capítulo serão apresentadas vulnerabilidades na conexão de um dispositivo via USB e respetivas vulnerabilidades experimentadas no Sistema Operativo *Android*. Um cenário de ataque e concretização do mesmo foi desenvolvido e o seu procedimento será seguidamente descrito. Existe também uma secção dedicada ao *Android Debugger Bridge*.

5.1 Vulnerabilidade na conexão de um dispositivo via USB

O protocolo USB [87, 88] é totalmente controlado por *software* e por isso é semelhante em todos os dispositivos *Android*. Existem dois sub-protocolos que são suportados pelo *Android*: (1) o *mass-storage class* (por exemplo um disco amovível USB) e (2) a *Android Debugger Bridge* (ADB).

Cada função possui um documento de classe de dispositivo associado que especifica o protocolo padrão para essa função. Isso permite que os *hosts* compatíveis com a classe e as funções periféricas possam interagir, sem o conhecimento detalhado do funcionamento de cada um. A conformidade da classe é potencialmente perigosa se o *host* e o periférico forem fornecidos por diferentes entidades.

A conexão USB não suporta a classe de rede, de áudio ou outras classes, tais como a *mass storage class* (MSC) ou o *Media Transfer Protocol* (MTP). Por omissão, o ADB está desativado, e o computador refere-se ao *Android* como um dispositivo de armazenamento em massa, com nenhuma das funções adicionais. Apenas o cartão SD do dispositivo é exposto através de USB, ao invés das suas partições de sistema e de dados. Quando a "depuração USB"(ADB) está habilitada, o dispositivo pode ser controlado com a mesma ferramenta "adb", que é fornecida no SDK do *Android*.

Esta ferramenta torna possível inserir e obter ficheiros de e para o dispositivo, instalar ficheiros APK, TCP e redirecionamentos UDP, etc.

Uma das mudanças de *kernel* específico do *Android Linux* é o *Paranoid-Network*. Normalmente, em sistemas *Linux*, um processo de espaço do utilizador pode abrir conexões de rede à vontade. No *Android*, uma aplicação de espaço do utilizador deve receber a permissão "INTERNET", a fim de fazer qualquer tipo de conexão de rede. Porque a mudança é ao nível do *kernel*, a estrutura da aplicação não participa na aplicação destas permissões. Na verdade, até mesmo as aplicações nativas estão sujeitas a esta definição. As definições da *Paranoid-Network* trabalham embutindo vários IDs de grupos *POSIX* no *kernel*. A aplicação deve ser de um dos grupos relevantes antes de ser autorizada a criar *sockets*.

Devido às vulnerabilidades que este processo apresentava, a *Google* a partir da versão 4.4.2 (KitKat), criou um processo de segurança para essa conexão: o *ADB pairing* (emparelhamento entre uma conexão USB e um *Android Debug Bridge* (ADB)). Certificando-se de que cada conexão USB possui um par de chaves RSA aceite pelo computador (*host*) que o dispositivo *Android* se irá conectar para, que assim, cada vez que o *smartphone Android* se tentar conectar com um novo *host*, tem que ter sido previamente aceite pelo mesmo.

Em seguida, apresentamos com detalhe a ferramenta ADB, que foi usada para implementar os cenários de ataque descritos na secção 5.3.

5.2 *Android Debugger Bridge*

O *Android Debugger Bridge* (ADB) [89] é uma ferramenta de linha de comandos versátil que permite a comunicação com uma instância de emulador ou com um dispositivo *Android* conectado. Facilita uma variedade de ações de dispositivo, como instalar ou depurar e fornece acesso a uma *Unix shell* que pode ser usada para executar diversos comandos num emulador ou em dispositivos conectados. É um programa cliente-servidor com três componentes:

1. **Um cliente**, que envia comandos. O cliente é executado no computador de desenvolvimento. É possível emitir um comando adb para invocar o cliente a partir de um terminal de linha de comandos.

2. **Um *daemon***, que executa comandos num dispositivo. Um *daemon* é executado como um processo de segundo plano em cada instância de emulador ou dispositivo.

3. **Um servidor**, que gere a comunicação entre o cliente e o *daemon*. O servidor é executado como um processo de segundo plano no computador de desenvolvimento.

Quando se inicia um cliente do adb, primeiro verifica-se se há um processo de servidor do adb em execução. Se não houver, esse processo é iniciado. Quando o servidor é iniciado, é vinculado à porta TCP 5037 local e escuta comandos enviados de clientes do adb — todos os clientes do adb usam a porta 5037 para se comunicar com o servidor do adb.

De seguida, o servidor configura conexões com todas as instâncias de emulador/-dispositivo em execução. Localiza as instâncias de emulador/dispositivo com um *scan* pelas portas ímpares no intervalo de 5555 a 5585, que é utilizado por emuladores/dispositivos. Onde o servidor encontrar um *daemon* do adb, configura uma conexão com a porta em questão. Cada instância de emulador/dispositivo adquire um par de portas sequenciais — uma porta par para conexões de console e uma porta ímpar para conexões do adb.

Quando o servidor configurar as conexões de todas as instâncias de emulador, será possível usar comandos do adb para aceder essas instâncias. Como o servidor gere as conexões com as instâncias de emulador/dispositivo e gere todos os comandos de vários clientes do adb, é possível controlar qualquer instância de emulador/dispositivo de qualquer cliente (ou de um *script*). Mais informações sobre o ADB podem ser encontradas no site oficial de onde esta informação também foi extraída [89].

5.3 Cenários de ataque

Nesta secção descreve-se o procedimento efetuado para tentar implementar uma solução que permita verificar a existência de vulnerabilidades na conexão de um dispositivo via USB. A solução passará por conectar um dispositivo *Android* via USB a um computador que contenha um *script*, que por sua vez consiga obter informação do dispositivo sem que este informe sobre esse acesso.

Foram identificados três cenários para a recuperação de informações privadas. Todos estão associados às conexões USB. A motivação por detrás destes cenários, prende-se com as características únicas de cada modelo: o primeiro cenário permite testar um modelo que tem a depuração USB disponível mas não ativa e está *rooted*, o segundo tem a depuração USB disponível mas não está *rooted* e o terceiro não tem a depuração USB disponível, não está *rooted* e possui uma versão do *Android*

com a tecnologia *ADB pairing*. Depois, um *script* foi implementado permitindo as seguintes ações:

1. Obter informações do dispositivo;
2. Lista de todos os *packages* instalados;
3. Cópia completa de todo o conteúdo do cartão SD;
4. Copiar um ficheiro para o dispositivo;
5. Instalar um ficheiro no dispositivo;
6. Executar uma aplicação no dispositivo;
7. Obter a lista de contactos;
8. Obter as mensagens;
9. Desbloquear o ecrã;
10. Superar o *ADB pairing*;
11. Efetuar o *Rooting* do dispositivo.

Um cenário de ataque e uma prova de conceito que se adapta às vulnerabilidades foram pesquisados. Esta combinação de cenário de ataque e prova de conceito também visou aumentar a consciencialização dos perigos associados à conexão USB.

Em relação aos dispositivos móveis foram testados três dispositivos, cada um com uma versão diferente do *Android*. Para cada versão foram também testadas condições específicas.

A depuração USB estar disponível é uma opção essencial para explorar as vulnerabilidades consideradas neste trabalho, uma vez que o único contacto do telefone com a máquina que contém o *Script* é feito através de um cabo USB. Muitos dos dispositivos podem ter esta opção ativa sem consciência dos perigos associados [90, 91].

Em primeiro lugar decidiu-se optar por um *Script* escrito para o Sistema Operativo **Windows 7 Home Premium** [92] e uma das suas ferramentas, a **Windows PowerShell ISE 5.0** [93].

O ficheiro vai ter extensão *.ps1* [94], que é o formato associado à **Windows PowerShell** da **Microsoft Corporation**.

Em relação aos cenários identificados, todos os três correspondem a vulnerabilidades que o *Script* é capaz de explorar. As versões usadas representam uma

percentagem reduzida dos dispositivos que visitaram recentemente a *Google Play Store* [95].

No entanto, provavelmente corresponderão a uma maior percentagem de entre os dispositivos efetivamente em uso, pois é de supor que os dispositivos mais antigos não efetuam uma utilização tão intensiva da *Google Play Store*.

- **Cenário de Ataque I:**
  Configuração:
  - Dispositivo: Samsung Galaxy Mini GT-S5570, ver Fig. 5.1
  - Versão do *Android*: 2.3.6 (*Gingerbread*)
  - *USB debugging* ***ON***
  - Dispositivo *rooted*

  Resultados Obtidos:
  1. Obter informações do dispositivo;
  2. Lista de todos os *packages* instalados;
  3. Cópia completa de todo o conteúdo do cartão SD;
  4. Copiar um ficheiro para o dispositivo;
  5. Instalar um ficheiro no dispositivo;
  6. Executar uma aplicação no dispositivo;
  7. Obter a lista de contactos;
  8. Obter as mensagens;

Figura 5.1: Samsung Galaxy Mini

Este é o cenário mais simples para atacar enquanto a depuração está ativada e o dispositivo está *rooted*. Este dispositivo possui uma característica específica,

que permite aumentar a vulnerabilidade da Depuração USB: é possível habilitar a conexão mas mantê-la inativa, dando uma falsa sensação de proteção ao seu utilizador, mas simultaneamente, deixando a porta aberta para um eventual ataque.

A característica específica *rooted* está presente num elevado número de dispositivos [96, 97] e tal pode suceder de várias formas: por opção do utilizador [98, 99], por sugestão de algo ou alguém ou simplesmente de forma involuntária [100].

Por esta via e com acesso *rooting* ao dispositivo, foi possível obter todo o tipo de informação desejada, comprometendo a segurança do mesmo.

- **Cenário de Ataque II:**
  Configuração:
  - Dispositivo: Sony Xperia Miro ST23i, ver Fig. 5.2
  - Versão do *Android*: 4.0.4 (*Ice Cream Sandwich*)
  - *USB debugging* ***ON***
  - Dispositivo *unrooted*

  Resultados Obtidos:

  1. Obter informações do dispositivo;
  2. Lista de todos os *packages* instalados;
  3. Cópia completa de todo o conteúdo do cartão SD;
  4. Copiar um ficheiro para o dispositivo;
  5. Instalar um ficheiro no dispositivo;
  6. Executar uma aplicação no dispositivo;
     Se a aplicação efetuar o *rooting* do dispositivo, então os três seguintes também são possíveis:
     a) Obter a lista de contactos;
     b) Obter as mensagens;
     c) Desbloquear o ecrã;

Este dispositivo não possui acesso *rooting*, o que limitaria à partida o alcance dos ataques. No entanto, ao habilitar a depuração USB, fica exposto a ataques ao cartão SD e a recolha de informação que permite comprometer o dispositivo com ataques que forcem esse acesso.

Figura 5.2: Sony Xperia Miro

- **Cenário de Ataque III:**
  Configuração:
  - Dispositivo: Aquaris E5 HD, ver Fig. 5.3
  - Versão do *Android*: 5.0 (*Lollipop*)
  - *USB debugging* ***OFF***
  - Dispositivo *unrooted*

Figura 5.3: Aquaris E5 HD

Este dispositivo não possui acesso *rooting* e sem que seja efetuado o *ADB pairing* não tem a depuração USB disponível. Este cenário foi testado por possuir uma versão superior à já mencionada 4.4.2 (KitKat), apenas para confirmar que o tipo de ataque implementado necessita sempre da depuração USB disponível, mas, uma vez concedido esse acesso, será possível comprometer qualquer tipo de dispositivo.

5.4 Criação do *Script*

O *Script* têm por base a auto-deteção de uma ligação USB, ou seja sempre que um dispositivo se conectar com o computador, será automaticamente detetado e o *Script* iniciará as suas funções. Uma vez identificado o dispositivo, serão efetuadas várias tentativas de obtenção de informação, através de comandos de *Windows PowerShell ISE 5.0* e/ou via *Android Debug Bridge* (ADB).

Todos estes processos são invisíveis à vitima e através dos mesmos, consegue-se aceder a todo o conteúdo do cartão SD e também a várias informações do dispositivo. Para ativar o funcionamento do ADB na *Windows PowerShell ISE 5.0* foi necessário efetuar o seguinte procedimento:

- Instalar o *chocolatey*[1]

  iwr https://chocolatey.org/install.ps1 -UseBasicParsing — iex

- Verificar se existem *upgrades* disponíveis

  choco upgrade chocolatey

- Utilizar a ferramenta para instalar o ADB

  choco install adb -y

- Verificar se existem *upgrades* disponíveis

  choco upgrade adb

5.5 Resultados obtidos

Em `http://y2u.be/TDgUgxgOt_o`, um *screencast* mostra o *script* a ser executado quando um *smartphone* é conectado ao computador *host*. Estes testes foram executados várias vezes e apresentaram sempre o mesmo resultado, para os cenários em questão, sem necessitar de alterar o *script* ou as configurações do dispositivo. Todos os resultados obtidos ou são mostrados no ecrã ou são guardados em ficheiros e pastas, ver Fig. 5.4.

1. **Obter informações do dispositivo**:

   O primeiro resultado obtido foi a obtenção da informação sobre o dispositivo, ver Fig. 5.8, que para além de ser mostrada no ecrã em maior detalhe, ver Fig.

[1] `https://chocolatey.org`

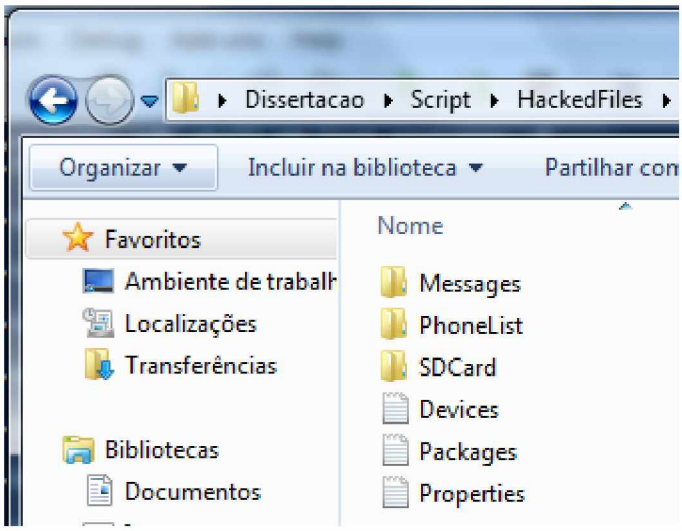

Figura 5.4: Pasta com o conteúdo obtido

5.6, também fica guardada num ficheiro de texto, ver Fig. 5.7. Foi possível obter a identificação do dispositivo, ver Fig. 5.5, bem como informação variada acerca do mesmo, tal como a *drive letter* que lhe é atribuída pelo Sistema Operativo, a versão do *Android*, o modelo entre outras.

```
USB drives detected:
   * SAMSUNG GT-S5570 Card USB Device
```

Figura 5.5: Lista dos dispositivos conectados e respetiva identificação mostrada no ecrã

2. **Lista de todos os *packages* instalados**:

   De seguida foi possível obter uma lista de todos os *packages* instalados no dispositivo, também no ecrã, ver Fig. 5.9, e em formato de ficheiro de texto, ver Fig. 5.10. A obtenção dessa lista foi então usada para obter o nome exato que o Sistema atribui a cada *package*. Isso foi útil para conhecer dados sobre a aplicação que foi instalada para executá-la. No item **Executar uma aplicação no dispositivo**, será explicada a utilidade desta informação.

3. **Cópia completa de todo o conteúdo do cartão SD**:

   Outro resultado obtido foi a cópia integral de todo o conteúdo do *SDCard* do dispositivo, ver Fig. 5.11 e Fig. 5.12. Devido ao pouco espaço de armazenamento que os dispositivos têm, a maioria dos utilizadores guarda uma grande parte do conteúdo pessoal no cartão SD. Este processo permite obter todos os ficheiros e pastas do cartão, tais como fotografias, filmes, etc.

```
2017-03-16T17:45:14  Beginning script...
2017-03-16T17:45:21  Event detected =  Device arrival
2017-03-16T17:45:21  Drive name =  I:
2017-03-16T17:45:21  Drive label =
List of devices attached
S55702f5a48e4   device

[ro.secure]: [1]

[ro.allow.mock.location]: [0]

[ro.debuggable]: [0]

[persist.service.adb.enable]: [1]

[ro.factorytest]: [0]

[ro.serialno]: []

[ro.bootmode]: [unknown]

[ro.baseband]: [unknown]

[ro.carrier]: [unknown]

[ro.bootloader]: [unknown]

[ro.hardware]: [gt-s5570]

[ro.revision]: [3]

[ro.emmc]: [0]

[ro.build.id]: [GINGERBREAD]

[ro.build.display.id]: [GINGERBREAD.XWKTL]

[ro.build.version.incremental]: [XWKTL]

[ro.build.version.sdk]: [10]

[ro.build.version.codename]: [REL]

[ro.build.version.release]: [2.3.6]
```

Figura 5.6: Lista dos dispositivos conectados e respetiva informação mostrada no ecrã

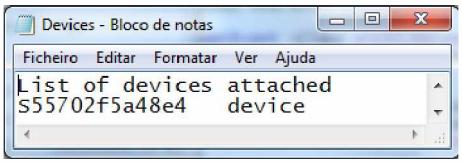

Figura 5.7: Lista dos dispositivos conectados guardada no ficheiro

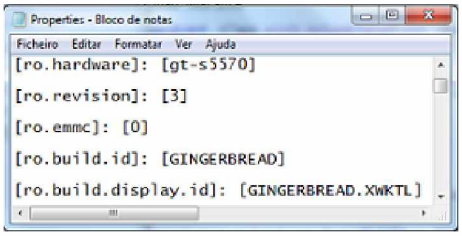

Figura 5.8: Lista da informação dos dispositivos conectados guardada no ficheiro

```
package:/system/framework/framework-res.apk=android
package:/system/app/TtsService.apk=android.tts
package:/system/app/Preconfig.apk=com.android.Preconfig
package:/system/app/BluetoothOpp.apk=com.android.bluetooth
package:/system/app/BluetoothTestMode.apk=com.android.bluetoothtest
package:/system/app/Browser.apk=com.android.browser
package:/system/app/Calculator.apk=com.android.calculator2
package:/system/app/Calendar.apk=com.android.calendar
package:/system/app/CertInstaller.apk=com.android.certinstaller
package:/system/app/Contacts.apk=com.android.contacts
package:/system/app/DefaultContainerService.apk=com.android.defcontainer
```

Figura 5.9: Lista dos *packages* instalados no dispositivo mostrada no ecrã

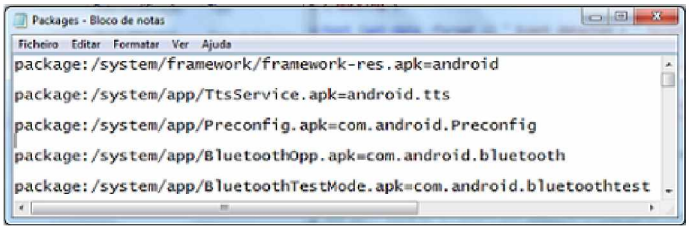

Figura 5.10: Lista dos *packages* instalados no dispositivo guardada no ficheiro

```
[ 98%] sdcard/Android/data/com.google.android.apps.maps/cache/cache_bd.m: 100%
[ 98%] sdcard/Android/data/com.google.android.apps.maps/cache/cache_vts.m: 100%
[ 99%] sdcard/Android/data/com.google.android.apps.maps/cache/cache_rgts.m: 100%
[ 99%] sdcard/Android/data/com.cooliris.media/cache/.gridcalc.res: 100%
[ 99%] sdcard/Android/data/com.cooliris.media/cache/local-image-thumbs/index: 100%
[ 99%] sdcard/Android/data/com.cooliris.media/cache/local-image-thumbs/chunk_0: 100%
[ 99%] sdcard/Android/data/com.cooliris.media/cache/local-meta-cache/chunk_0: 100%
[ 99%] sdcard/Android/data/com.cooliris.media/cache/local-meta-cache/index: 100%
[ 99%] sdcard/Android/data/com.cooliris.media/cache/local-album-cache/chunk_0: 100%
[ 99%] sdcard/Android/data/com.cooliris.media/cache/local-album-cache/index: 100%
[ 99%] sdcard/LOST.DIR/14: 100%
sdcard/: 86 files pulled. 0 files skipped. 2.2 MB/s (11292550 bytes in 4.789s)
```

Figura 5.11: Lista do conteúdo do *SDCard* do dispositivo mostrada no ecrã

4. **Copiar um ficheiro para o dispositivo**:

   Depois de conseguir obter o conteúdo do *SDCard*, foi possível copiar um ficheiro (neste caso uma aplicação que permite efetuar o *Rooting* do *Android*) [101] para o dispositivo e verificar que ele estava no respetivo cartão, ver Fig. 5.13, Fig. 5.14 e Fig. 5.15. Este processo permite colocar no cartão qualquer tipo de aplicação. Caso seja um *mallware* ou algo do género, é possível comprometer todo o dispositivo.

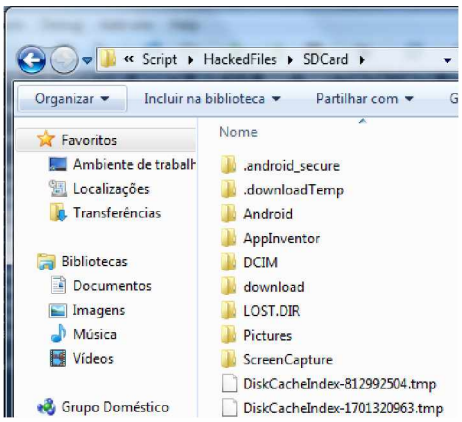

Figura 5.12: Lista do conteúdo do *SDCard* do dispositivo guardada no ficheiro

```
[  7%] /sdcard/z4root.1.1.0.apk
[ 14%] /sdcard/z4root.1.1.0.apk
[ 21%] /sdcard/z4root.1.1.0.apk
[ 28%] /sdcard/z4root.1.1.0.apk
[ 35%] /sdcard/z4root.1.1.0.apk
[ 42%] /sdcard/z4root.1.1.0.apk
[ 49%] /sdcard/z4root.1.1.0.apk
[ 56%] /sdcard/z4root.1.1.0.apk
[ 63%] /sdcard/z4root.1.1.0.apk
[ 70%] /sdcard/z4root.1.1.0.apk
[ 77%] /sdcard/z4root.1.1.0.apk
[ 84%] /sdcard/z4root.1.1.0.apk
[ 91%] /sdcard/z4root.1.1.0.apk
[ 98%] /sdcard/z4root.1.1.0.apk
[100%] /sdcard/z4root.1.1.0.apk
```

Figura 5.13: Processo de cópia do ficheiro para o dispositivo mostrado no ecrã

```
pull: building file list...
[  0%] sdcard/z4root.1.1.0.apk: 7%
[  1%] sdcard/z4root.1.1.0.apk: 14%
[  1%] sdcard/z4root.1.1.0.apk: 21%
[  2%] sdcard/z4root.1.1.0.apk: 28%
[  2%] sdcard/z4root.1.1.0.apk: 35%
[  3%] sdcard/z4root.1.1.0.apk: 42%
[  4%] sdcard/z4root.1.1.0.apk: 49%
[  4%] sdcard/z4root.1.1.0.apk: 56%
[  5%] sdcard/z4root.1.1.0.apk: 63%
[  5%] sdcard/z4root.1.1.0.apk: 70%
[  6%] sdcard/z4root.1.1.0.apk: 77%
[  6%] sdcard/z4root.1.1.0.apk: 84%
[  7%] sdcard/z4root.1.1.0.apk: 91%
[  8%] sdcard/z4root.1.1.0.apk: 98%
[  8%] sdcard/z4root.1.1.0.apk: 100%
```

Figura 5.14: Lista do inicio do conteúdo do *SDCard* do dispositivo mostrado no ecrã, onde se pode verificar a existência do novo ficheiro

5. **Instalar um ficheiro no dispositivo**:

   Uma vez que o ficheiro copiado para o dispositivo é uma aplicação é possível efetuar a instalar a mesma, ver Fig. 5.16 e Fig. 5.17. Este processo é extremamente perigoso para o *Android*, caso a aplicação que está a ser instalada tenha capacidade para corromper o mesmo, das mais variadas formas, como

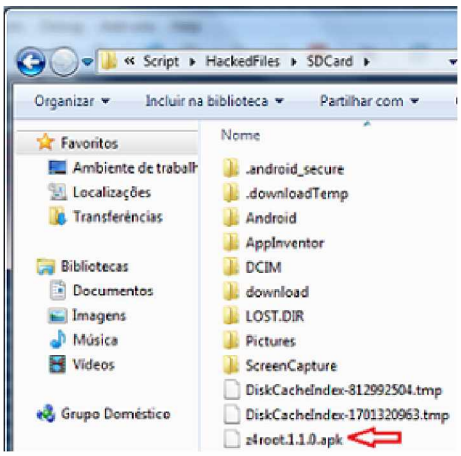

Figura 5.15: Conteúdo da pasta ..\HackedFiles\SDCard, onde se pode verificar a existência do novo ficheiro

por exemplo um *mallware*.

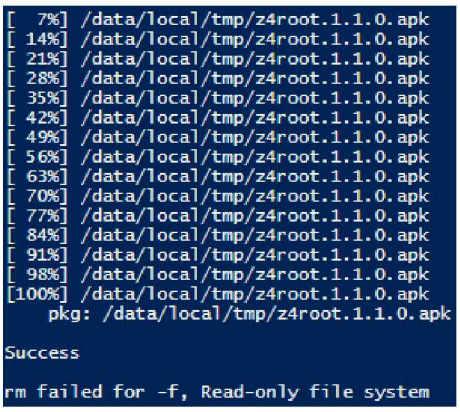

Figura 5.16: Processo de instalação da aplicação para o dispositivo mostrado no ecrã

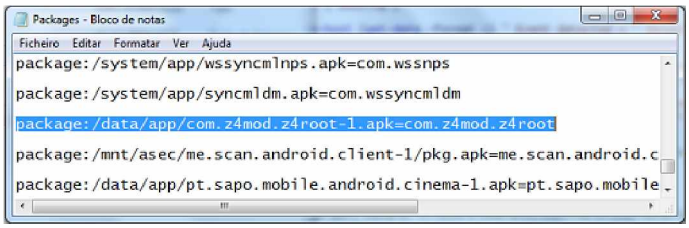

Figura 5.17: Lista dos *packages* instalados no dispositivos guardada no ficheiro, onde se pode verificar a existência da nova aplicação instalada

6. **Executar uma aplicação no dispositivo**:

   Depois de instalar a aplicação no dispositivo é ainda possível executá-la, ver Fig. 5.18 e Fig. 5.19, o que no caso de ser uma aplicação maliciosa levanta

sérios problemas de segurança. Este processo acontece, devido ao facto de ser possível verificar uma listagem de todas as aplicações instaladas e respetiva informação. O conteúdo desta lista, permite saber o nome exato que o Sistema "atribui"a cada *package*, sendo depois muito útil para obter o caminho exato para executar o comando completo para executar a aplicação.

```
Events injected: 1

## Network stats: elapsed time=51ms (0ms mobile, 0ms wifi, 51ms not connected)
```

Figura 5.18: Processo de execução da aplicação no dispositivo mostrado no ecrã

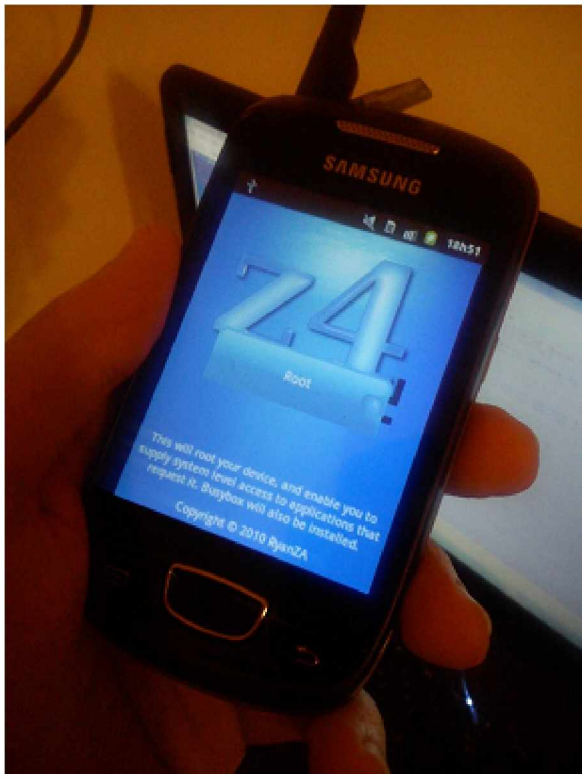

Figura 5.19: Imagem da execução da aplicação no dispositivo

7. **Obter a lista de contactos**:

   Um dos principais resultados foi a obtenção de informação privada do utilizador do dispositivo. Neste caso foi copiado o conteúdo da lista de contactos, ver Fig. 5.20, Fig. 5.21 e Fig. 5.22. Este processo só é possível se o *rooting* do dispositivo tiver sido feito, mas como é possível copiar, instalar e executar aplicações apenas com a depuração USB ligada, é possível fazer isto de forma

invisível à vitima e obter este tipo de informação, que neste caso necessitou de alteração de permissões de utilizador, que permitiram a obtenção da desejada lista em formato SQL e *plain text*.

```
[ 60%] /data/data/com.android.providers.contacts/databases/contacts2.db
[100%] /data/data/com.android.providers.contacts/databases/contacts2.db
```

Figura 5.20: Captura da lista de contactos existentes no dispositivo mostrado no ecrã

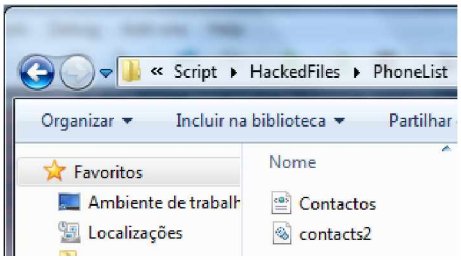

Figura 5.21: Conteúdo da pasta ..\HackedFiles\PhoneList, onde se pode verificar a existência dos ficheiros de contactos

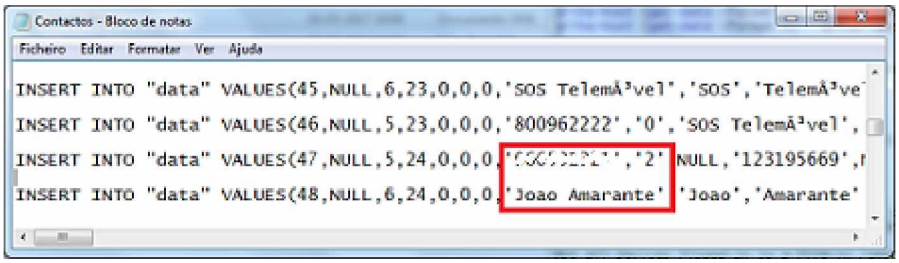

Figura 5.22: Lista dos contactos existentes no dispositivo guardada no ficheiro, onde se pode verificar a existência do um contacto

8. **Obter as mensagens**:

   Outro caso semelhante de obtenção de informação privada foi o conteúdo de todas as SMS MMS existentes no dispositivo, ver Fig. 5.23, Fig. 5.24 e Fig. 5.25. Utilizando o processo anteriormente descrito foi também possível obter a informação da respetiva lista em formato SQL e *plain text*.

```
[100%] /data/data/com.android.providers.telephony/databases/mmssms.db
```

Figura 5.23: Captura das mensagens existentes no dispositivo mostrado no ecrã

9. **Desbloquear o ecrã**:

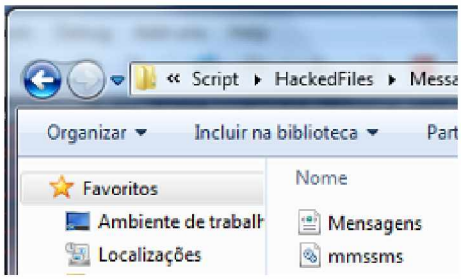

Figura 5.24: Conteúdo da pasta ..\HackedFiles\Messages, onde se pode verificar a existência dos ficheiros de mensagens

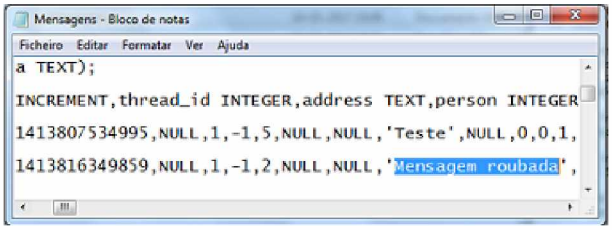

Figura 5.25: Lista dos mensagens existentes no dispositivo guardada no ficheiro, onde se pode verificar a existência do uma mensagem

Nos cenários identificados e implementados é possível ignorar/desativar o desbloqueio de *patterns* de ecrã no *Android* por meio de comandos do ADB, mas apenas se duas condições especiais forem atendidas: (1) o dispositivo estiver *rooted* e (2) o código PIN (ou de desbloqueio) for conhecido. Isso ocorre porque é necessário remover ou atualizar o ficheiro do sistema que contém a chave de bloqueio do ecrã, o que só é possível alterando as permissões de acesso do mesmo, conforme descrito anteriormente. Para finalizar é necessário reiniciar o dispositivo, exigindo o código PIN para o efeito.

10. **Superar o *ADB pairing***:

    É possível eliminar esse aprimoramento de segurança, se o dispositivo estiver *rooted*, removendo o ficheiro do sistema que contém a chave de emparelhamento ADB (*ADB pairing*). No entanto, um primeiro acesso ao dispositivo deve ser concedido, de modo que a que seja possível eliminar esse problema permanentemente.

11. **Efetuar o *Rooting* do dispositivo**:

    É possível efetuar o *Rooting* de um dispositivo, simplesmente copiando, instalando e executando uma aplicação (ver Fig. 5.19).

Capítulo 6

Conclusão

Nesta dissertação, foram apresentadas diversas Vulnerabilidades nas conexões via USB em Dispositivos com o Sistema *Android*. Na pesquisa de vulnerabilidades, concluiu-se que sempre que a depuração USB está disponível, o dispositivo está comprometido.

No modelo específico **Samsung Galaxy Mini GT-S5570** com a versão 2.3.6 **Gingerbread** existe ainda uma particularidade: o *USB Debugging* está disponível mas em estado "não ativo", o que permite uma falsa ilusão ao utilizador de estar protegido quando na verdade não está.

Verificou-se também que apesar de existir *software de proteção* como por exemplo um anti-vírus, ele não impede o processo de instalação de uma aplicação potencialmente maliciosa, apenas lança o alerta de que determinada aplicação poderá ser prejudicial, como no caso da aplicação **z4root.1.1.0**, utilizada na implementação de vulnerabilidades.

Foi fornecido um cenário de ataque prático onde vários ataques foram realizados e onde, os utilizadores com menos conhecimentos, que são provavelmente os mais comuns, podem ser efetivamente mais afetados.

Como as vulnerabilidades dependem do dispositivo e da versão do Sistema Operativo, a prova de conceito foi construída para funcionar em várias versões do Android e obter o máximo de informação. Foi demonstrado que para as versões 2.3.6 *Gingerbread* e 4.0.4 *Ice Cream Sandwich* do *Android*, após a ligação de um dispositivo móvel a um computador comprometido, é possível obter essa informação utilizando ferramentas facilmente disponíveis. A vulnerabilidade não é o *Android Debugger Bridge* (ADB) em si, mas sim ser possível utilizá-lo para obter informação sem que o utilizador tenha conhecimento.

A única forma de prevenir os ataques é nunca disponibilizar o *USB Debugging*.

Uma vez dada essa porta de acesso, todo o dispositivo pode ficar comprometido em segundos e de forma completamente invisível à vitima.

Ficou em aberto a possibilidade de conseguir ativar o *USB Debugging* através de conexão USB, permitindo aumentar, ainda mais, o grau de vulnerabilidade existente neste procedimento. Questões forenses ou de outro âmbito não foram consideradas, uma vez que o foco da dissertação foi apenas os perigos resultantes da ligação USB.

Outra tarefa futura será a implementação de novos cenários de ataque com sucesso, nomeadamente nas versões mais recentes do *Android*, ultrapassando a questão do *ADB pairing*.

Seria interessante também, fazer uma experiência social para contabilizar quantos dispositivos se conseguiriam atacar por hora, por dia ou mesmo semana, num local onde fosse oferecida a possibilidade de carregar gratuitamente a bateria dos dispositivos. Tal estudo poderia ser feito em diversos lugares, diferentes ambientes, de forma a determinar em que situação as pessoas são mais propensas a serem afetadas.

Bibliografia

[1] A. O. S. Project, "Android software stack," https://source.android.com/license.html, 2017, https://source.android.com/security/ [Online; accessed 26-March-2017]. (citado nas págs. xi e 9)

[2] K. Parmar, "In depth: Android package manager and package installer," https://dzone.com/mobile-app-developer-tutorials-tools-news, 2013, https://dzone.com/articles/depth-android-package-manager [Online; accessed 26-March-2017]. (citado nas págs. xi e 10)

[3] L. Huawei Technologies Co., "Android ril integration guide," Copyright © Huawei Technologies Co., Ltd. 2014. All rights reserved, 2014, https://www.paoli.cz/out/media/HUAWEI_Module_Android_RIL_Integration_Guide-V100R001_V3_4.pdf [Online; accessed 26-March-2017]. (citado nas págs. xi e 29)

[4] U. C. at Computex, "Usb device class specifications," 2017, http://www.usb.org/developers/docs/devclass_docs [Online; accessed 28-March-2017]. (citado na pág. 1)

[5] Google, "android," 2017, http://www.android.com [accessed 28-March-2017]. [*Online*]. Disponível: http://www.android.com (citado na pág. 2)

[6] S. Cope, "Bluetooth on android-what is it and how it works," 2017, http://www.stevesandroidguide.com/bluetooth [Online; accessed 28-March-2017]. (citado na pág. 2)

[7] D. Kushner, "The real story of stuxnet," *IEEE Spectrum*, vol. 50, n. 3, pp. 48–53, March 2013. [*Online*]. Disponível: http://ieeexplore.ieee.org/document/6471059/ (citado na pág. 2)

[8] M. M. P. Center, "Centralized information about the conficker worm," 2009, https://blogs.technet.microsoft.com/mmpc/2009/01/22/centralized-

information-about-the-conficker-worm [Online; accessed 28-March-2017]. (citado na pág. 3)

[9] Statista, "Number of smartphone users worldwide from 2014 to 2020 (in billions)," 2017, https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ [accessed 28-March-2017]. [*Online*]. Disponível: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (citado na pág. 3)

[10] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, e M. Bailey, "Users really do plug in usb drives they find," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 306–319. (citado na pág. 4)

[11] J. Gozalvez, "First google's android phone launched [mobile radio]," *IEEE Vehicular Technology Magazine*, vol. 3, n. 4, pp. 3–69, December 2008. [*Online*]. Disponível: http://ieeexplore.ieee.org/abstract/document/4799795/ (citado na pág. 7)

[12] R. Sandberg, *The business of Android Apps development : making and marketing Apps that succeed on Google Play, Amazon App Store and more.* New York: Apress, 2013. (citado na pág. 7)

[13] M. Schimmer, P. D. G. Müller-Stewens, e P. Spoonland, "The battle between apple, microsoft and google," University of St.Gallen Case Study - reference no 310-245-1, 2010, http://www.ifb.unisg.ch/~/media/internet/content/dateien/instituteundcenters/ifb/lehre/lehrbuecher%20und%20fallstudien/case%20studies/the%20battle_310-245-1s_inspection%20copy_c_en.pdf [Online; accessed 26-March-2017]. (citado na pág. 7)

[14] P. Carvalho, "Estes são os níveis de market share de novembro de acordo com a kantar," 2017, https://4gnews.pt/estes-sao-os-niveis-de-market-share-de-novembro-de-acordo-com-a-kantar [Online; accessed 28-March-2017]. (citado na pág. 8)

[15] J. Gosling, B. Joy, G. Steele, G. Bracha, e A. Buckley, "The java language specification - java se 8 edition," Oracle, Oracle Specification, February 2015, https://docs.oracle.com/javase/specs/jls/se8/html/index.html. [*Online*]. Disponível: https://docs.oracle.com/javase/specs/jls/se8/html/index.html (citado na pág. 8)

[16] Wikipedia, "Linux kernel — wikipedia, the free encyclopedia," 2017, [accessed 25-March-2017]. [*Online*]. Disponível: https://en.wikipedia.org/w/index.php?title=Linux_kernel&oldid=771603459 (citado na pág. 8)

[17] ——, "Linux — wikipedia, the free encyclopedia," 2017, [accessed 25-March-2017]. [*Online*]. Disponível: https://en.wikipedia.org/w/index.php?title=Linux&oldid=771832135 (citado na pág. 8)

[18] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, e C. Glezer, "Google android: A comprehensive security assessment," *IEEE Security & Privacy Magazine*, vol. 8, n. 2, pp. 35–44, March 2010. (citado na pág. 11)

[19] A. Shabtai, Y. Fledel, e Y. Elovici, "Securing android-powered mobile devices using selinux," *IEEE Security Privacy*, vol. 8, n. 3, pp. 36–44, May 2010. (citado na pág. 11)

[20] T. Bläsing, L. Batyuk, A. D. Schmidt, S. A. Camtepe, e S. Albayrak, "An android application sandbox system for suspicious software detection," in *2010 5th International Conference on Malicious and Unwanted Software*, Oct 2010, pp. 55–62. (citado na pág. 11)

[21] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, e S. Dolev, "Google android: A state-of-the-art review of security mechanisms," *CoRR*, vol. abs/0912.5101, 2009. [*Online*]. Disponível: http://arxiv.org/abs/0912.5101 (citado na pág. 11)

[22] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, e C. Glezer, "Google android: A comprehensive security assessment," *IEEE Security & Privacy Magazine*, vol. 8, n. 2, pp. 35–44, mar 2010. [*Online*]. Disponível: https://doi.org/10.1109/msp.2010.2 (citado na pág. 13)

[23] S. K. Singh, B. Mishra, e P. Gera, "A privacy enhanced security framework for android users," in *2015 5th International Conference on IT Convergence and Security (ICITCS)*, Aug 2015, pp. 1–6. (citado na pág. 15)

[24] B. Andow e H. Wang, "A distributed android security framework," in *Proceedings of the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, Dec 2015, pp. 1045–1052. (citado na pág. 15)

[25] TutorialsPoint, "Android - architecture," 2017, [accessed 27-March-2017]. [*Online*]. Disponível: https://www.tutorialspoint.com/android/android_architecture.htm (citado na pág. 15)

[26] S. Butt, V. Ganapathy, M. M. Swift, e C. C. Chang, "Protecting commodity operating system kernels from vulnerable device drivers," in *2009 Annual Computer Security Applications Conference*, Dec 2009, pp. 301–310. (citado na pág. 16)

[27] H. Wiechman, K. Chalmers, e C. Maupin, "Mainline linux™ ensures stability and innovation," 2017, [accessed 27-March-2017]. [*Online*]. Disponível: http://www.ti.com/lit/wp/spry259/spry259.pdf (citado na pág. 16)

[28] A. Shabtai, Y. Fledel, e Y. Elovici, "Detecting malicious applications on android using anomaly detection," *The 25th Annual Computer Security Applications Conference (ACSAC'09)*, pp. 36 – 44, December 2009. [*Online*]. Disponível: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5342408 (citado nas págs. 17, 22, 50 e 60)

[29] A. D. Project, "Android security overview - permissions," 2017, [accessed 27-March-2017]. [*Online*]. Disponível: https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/security/permissions.html (citado na pág. 18)

[30] Z. Xu, "Android installer hijacking," 2015, [accessed 27-March-2017]. [*Online*]. Disponível: http://researchcenter.paloaltonetworks.com/2015/03/android-installer-hijacking-vulnerability-could-expose-android-users-to-malware/ (citado na pág. 19)

[31] L. Chen, "Attacking webkit applications by exploiting memory corruption bugs," 2014, [accessed 27-March-2017]. [*Online*]. Disponível: https://cansecwest.com/slides/2015/Liang_CanSecWest2015.pdf (citado na pág. 19)

[32] M. Khari, Sonam, Vaishali, e M. Kumar, "Comprehensive study of web application attacks and classification," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, March 2016, pp. 2159–2164. (citado na pág. 19)

[33] U. S. C. E. R. Team, "Vulnerability summary for the week of january 30, 2017," 2017, [accessed 27-March-2017]. [*Online*]. Disponível: https://www.us-cert.gov/ncas/bulletins/SB17-037 (citado na pág. 19)

[34] X. Wang, X. Li, e W. Wen, "Wlcleaner: Reducing energy waste caused by wakelock bugs at runtime," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, Aug 2014, pp. 429–434. (citado na pág. 20)

[35] Wikipedia, "Oracle america, inc. v. google, inc." 2017, [accessed 28-March-2017]. [*Online*]. Disponível: https://en.wikipedia.org/wiki/Oracle_America,_Inc._v._Google,_Inc. (citado na pág. 21)

[36] J. Kastrenakes, "Google plans to start blocking flash in chrome this year," 2016, [accessed 28-March-2017]. [*Online*]. Disponível: http://www.theverge.com/2016/5/15/11679394/chrome-to-block-flash-later-2016 (citado na pág. 21)

[37] G. Lawton, "Is it finally time to worry about mobile malware?" *Computer*, vol. 41, n. 5, pp. 12–14, may 2008. [*Online*]. Disponível: https://doi.org/10.1109/mc.2008.159 (citado na pág. 23)

[38] Q. Do, B. Martini, e K.-K. R. Choo, "Exfiltrating data from android devices," *Comput. Secur.*, vol. 48, n. C, pp. 74–91, Fevereiro 2015. [*Online*]. Disponível: http://dx.doi.org/10.1016/j.cose.2014.10.016 (citado na pág. 24)

[39] Y. Moreau, H. Verrelst, e J. Vandewalle, *Detection of mobile phone fraud using supervised neural networks: A first prototype.* Proceedings of the 7th international Conference on Artificial Neural Networks, 1997. (citado na pág. 24)

[40] D. Samfat e R. Molva, "Idamn: an intrusion detection architecture for mobile networks," *IEEE Journal on Selected Areas in Communications*, vol. 15, n. 7, pp. 1373 – 1380, September 1997. [*Online*]. Disponível: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=622919 (citado na pág. 24)

[41] O. Sheyner, J. Haines, S. Jha, R. Lippmann, e J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, ser. SP '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 273–. [*Online*]. Disponível: http://dl.acm.org/citation.cfm?id=829514.830526 (citado na pág. 24)

[42] C. Guo, H. Wang, e W. Zhu, "Smart-phone attacks and defenses," 2004, http://conferences.sigcomm.org/hotnets/2004/HotNets-III[Online; accessed 28-March-2017]. (citado na pág. 25)

[43] T. S. Yap e H. T. Ewe, "A mobile phone malicious software detection model with behavior checker," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 57– 65, June 2005. [*Online*]. Disponível: http://dl.acm.org/citation.cfm?id=2130373 (citado na pág. 25)

[44] D. e. a. Nash, "Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices," *Pervasive Computing and Communications Workshops*, pp. 141 – 145, March 2005. [*Online*]. Disponível: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1392818 (citado na pág. 25)

[45] G. Jacoby e H. Davis, "Battery-based intrusion detection," *Global Telecommunications Conference*, pp. 2250 – 2255, November 2006. [*Online*]. Disponível: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1378409 (citado na pág. 25)

[46] M. Miettinen, P. Hälönen, e K. Hätönen, "Host-based intrusion detection for advanced mobile devices," *Proceedings of the 20th international conference on advanced information networking and applications*, pp. 72 – 76, April 2006. [*Online*]. Disponível: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1620356 (citado na pág. 25)

[47] J. Cheng, S. Wong, H. Yang, e S. Lu, "Smartsiren: virus detection and alert for smartphones," *Proceedings of the 5th international Conference on Mobile Systems, Applications and Services*, pp. 258 – 271, June 2007. [*Online*]. Disponível: http://dl.acm.org/citation.cfm?doid=1247660.1247690 (citado na pág. 25)

[48] A. Bose, X. Hu, K. Shin, e T. Park, "Behavioral detection of malware on mobile handsets," *Proceeding of the 6th international Conference on Mobile Systems, Applications, and Services*, pp. 225 – 238, June 2008. [*Online*]. Disponível: http://dl.acm.org/citation.cfm?doid=1378600.1378626 (citado na pág. 26)

[49] H. Kim, J. Smith, e K. Shin, "Detecting energy-greedy anomalies and mobile malware variants," *Proceeding of the 6th international Conference on Mobile Systems, Applications, and Services*, pp. 239 – 252, June 2008. [*Online*]. Disponível: http://dl.acm.org/citation.cfm?doid=1378600.1378627 (citado na pág. 26)

[50] T. e. a. Buennemeyer, "Mobile device profiling and intrusion detection using smart batteries," *International Conference on System Sciences*, January 2008. [*Online*]. Disponível: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4439001 (citado na pág. 26)

[51] A. Schmidt, F. Peters, F. Lamour, C. Scheel, S. Camtepe, e S. Albayrak, "Monitoring smartphones for anomaly detection," *Mobile Networks and Applications*, pp. 92 – 106, February 2009. [*Online*]. Disponível: http://dl.acm.org/citation.cfm?id=1361542 (citado nas págs. 26, 49 e 60)

[52] S. Hwang, S. Cho, e S. Park, "Keystroke dynamics-based authentication for mobile devices," *Computer and Security*, pp. 85 – 93, March 2009. [*Online*]. Disponível: http://www.sciencedirect.com/science/article/pii/S0167404808000965 (citado na pág. 27)

[53] D. Damopoulos, S. A. Menesidou, G. Kambourakis, M. Papadaki, N. Clarke, e S. Gritzalis, "Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers," *Security and Communication Networks*, vol. 5, n. 1, pp. 3–14, jun 2011. [*Online*]. Disponível: https://doi.org/10.1002/sec.341 (citado na pág. 27)

[54] K. Hamandi, A. Chehab, I. H. Elhajj, e A. Kayssi, "Android sms malware: Vulnerability and mitigation," in *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, March 2013, pp. 1004–1009. (citado na pág. 27)

[55] P. Colp, J. Zhang, J. Gleeson, S. Suneja, E. de Lara, H. Raj, S. Saroiu, e A. Wolman, "Protecting data on smartphones and tablets from memory attacks," *SIGPLAN Not.*, vol. 50, n. 4, pp. 177–189, Março 2015. [*Online*]. Disponível: http://doi.acm.org/10.1145/2775054.2694380 (citado na pág. 27)

[56] Z. Wang e A. Stavrou, "Exploiting smart-phone usb connectivity for fun and profit," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 357–366. [*Online*]. Disponível: http://doi.acm.org/10.1145/1920261.1920314 (citado na pág. 28)

[57] Z. Wang, R. Johnson, R. Murmuria, e A. Stavrou, "Exposing security risks for commercial mobile devices," in *Proceedings of the 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security: Computer Network Security*, ser. MMM-ACNS'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 3–21. [*Online*]. Disponível: http://dx.doi.org/10.1007/978-3-642-33704-8_2 (citado na pág. 28)

[58] M. Xu, "Security enhancement of secure USB debugging in Android system," Dissertação de mestrado, University of Toledo, USA, 2014, in http://utdr.utoledo.edu/theses-dissertations. (citado na pág. 28)

[59] M. Xu, W. Sun, e M. Alam, "Security enhancement of secure USB debugging in android system," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2015, pp. 134–139. (citado na pág. 28)

[60] A. Pereira, M. Correia, e P. Brandão, *USB Connection Vulnerabilities on Android Smartphones: Default and Vendors' Customizations.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 19–32. [*Online*]. Disponível: http://dx.doi.org/10.1007/978-3-662-44885-4_2 (citado nas págs. 28 e 29)

[61] Wikipedia, "Samsung kies — wikipedia, the free encyclopedia," 2017, [accessed 26-March-2017]. [*Online*]. Disponível: https://en.wikipedia.org/wiki/Samsung_Kies (citado na pág. 29)

[62] M. Neugschwandtner, A. Beitler, e A. Kurmus, "A transparent defense against USB eavesdropping attacks," in *Proceedings of the 9th European Workshop on System Security*, ser. EuroSec '16. New York, NY, USA: ACM, 2016, pp. 6:1–6:6. [*Online*]. Disponível: http://doi.acm.org/10.1145/2905760.2905765 (citado na pág. 30)

[63] Hak5, "Android hacking with the USB rubber ducky," 2012, https://www.hak5.org/episodes/hak5-1216 [Online; accessed 28-March-2017]. (citado na pág. 30)

[64] D. J. Tian, A. Bates, e K. Butler, "Defending against malicious USB firmware with goodusb," in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC 2015. New York, NY, USA: ACM, 2015, pp. 261–270. [*Online*]. Disponível: http://doi.acm.org/10.1145/2818000.2818040 (citado na pág. 30)

[65] D. J. Tian, N. Scaife, A. Bates, K. Butler, e P. Traynor, "Making usb great again with usbfilter," in *25th USENIX Security Symposium (USENIX Security 16).* Austin, TX: USENIX Association, 2016, pp. 415–430. [*Online*]. Disponível: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tian (citado na pág. 30)

[66] A. Zhukov, "Turning a regular usb flash drive into a usb rubber ducky," 2017, https://hackmag.com/author/zhukov/ [Online; accessed 28-March-2017]. (citado na pág. 30)

[67] K. Nohl e J. Lell, "Badusb - on accessories that turn evil," 2017, http://www.blackhat.com/us-14/briefings.html#badusb-on-accessories-that-turn-evil [Online; accessed 28-March-2017]. (citado na pág. 30)

[68] G. Ozuru, "Payload-teensy (badusb) like a rubber ducky," 2017, https://github.com/Screetsec/Pateensy [Online; accessed 28-March-2017]. (citado na pág. 30)

[69] G. brandonlw, "Making badusb work for you – derbycon," 2014, https://adamcaudill.com/2014/10/02/making-badusb-work-for-you-derbycon [Online; accessed 28-March-2017]. (citado na pág. 33)

[70] G. Ozuru, "Phison 2251-03 (2303) custom firmware and existing firmware patches," 2017, https://github.com/brandonlw/Psychson [Online; accessed 28-March-2017]. (citado na pág. 33)

[71] G. brandonlw, "Useful links," https://github.com/brandonlw/Psychson/wiki/Useful-Links, 2017, [Online; accessed 28-March-2017]. (citado na pág. 34)

[72] A. Software, "Flash drive information extractor," 2017, http://www.antspec.com/usbflashinfo/index.php [Online; accessed 28-March-2017]. (citado na pág. 35)

[73] D. Toolkit, "Payload generation," 2017, https://ducktoolkit.com/payload [Online; accessed 28-March-2017]. (citado na pág. 35)

[74] G. hak5darren, "Payloads," 2017, https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads [Online; accessed 28-March-2017]. (citado na pág. 35)

[75] ——, "Downloads," 2014, https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Downloads [Online; accessed 28-March-2017]. (citado na pág. 35)

[76] P. E. Corp, "Phison-controller," 2017, http://www.usbdev.ru/files/phison [Online; accessed 28-March-2017]. (citado na pág. 37)

[77] A. Community, "Security enhancements," 2017, https://source.android.com/security/enhancements/index.html [Online; accessed 28-March-2017]. (citado na pág. 40)

[78] ——, "Sslsocket," https://developer.android.com/reference/javax/net/ssl/SSLSocket.html, 2017, [Online; accessed 28-March-2017]. (citado na pág. 46)

[79] (2016) Clamav is an open source antivirus engine for detecting trojans, viruses, malware and other malicious threats. [*Online*]. Disponível: http://www.clamav.net (citado na pág. 48)

[80] (2016) Mobile security solutions. [*Online*]. Disponível: http://www.trendmicro.com/us/enterprise/product-security/mobile-security/index.html (citado na pág. 50)

[81] (2016) Norton security deluxe our best protection for any way you choose to connect. [*Online*]. Disponível: https://us.norton.com (citado na pág. 50)

[82] A. Shabtai, D. Mimran, e Y. Elovici, "Evaluation of security solutions for android systems," *CoRR*, vol. abs/1502.04870, 2015. [*Online*]. Disponível: http://arxiv.org/abs/1502.04870 (citado na pág. 52)

[83] G. Developers, "Storing application data," 2017, https://developers.google.com/drive/android/appfolder [Online; accessed 28-March-2017]. (citado na pág. 57)

[84] ——, "Low ram configuration," https://source.android.com/devices/tech/config/low-ram, 2017, [Online; accessed 28-March-2017]. (citado na pág. 57)

[85] (2016) security and data integrity tool useful for monitoring and alerting on specific file change(s) on a range of systems. [*Online*]. Disponível: https://sourceforge.net/projects/tripwire (citado na pág. 58)

[86] G. Developers, "Public constructors - dexfile," 2017, https://developer.android.com/reference/dalvik/system/DexFile.html [Online; accessed 28-March-2017]. (citado na pág. 59)

[87] D. Noyes, H. Liu, e P. Fortier, "Security analysis and improvement of usb technology," in *2016 IEEE Symposium on Technologies for Homeland Security (HST)*, May 2016, pp. 1–3. (citado na pág. 61)

[88] H. Jeong, Y. Choi, W. Jeon, F. Yang, Y. Lee, S. Kim, e D. Won, "Vulnerability analysis of secure usb flash drives," in *2007 IEEE International Workshop on Memory Technology, Design and Testing*, Dec 2007, pp. 61–64. (citado na pág. 61)

[89] Android Open Source project, "Android debug bridge," 2017, https://developer.android.com/studio/command-line/adb.html [accessed 28-March-2017]. [*Online*]. Disponível: https://developer.android.com/studio/command-line/adb.html (citado nas págs. 62 e 63)

[90] D. Thomas, "How to enable developer options and usb debugging," 2015, https://android.gadgethacks.com/how-to/android-basics-enable-developer-options-usb-debugging-0161948 [Online; accessed 28-March-2017]. (citado na pág. 64)

[91] M. Egan, "How to get developer options on android," 2016, http://www.pcadvisor.co.uk/how-to/google-android/34-useful-things-you-can-do-in-android-developer-options-new-3590299 [Online; accessed 28-March-2017]. (citado na pág. 64)

[92] Microsoft, "Documentation for windows 7 and windows server 2008 r2 service pack 1 (kb976932)," 2017, [accessed 28-March-2017]. [*Online*]. Disponível: https://www.microsoft.com/en-us/download/details.aspx?id=269 (citado na pág. 64)

[93] ——, "What's new with powershell," 2017, [accessed 28-March-2017]. [*Online*]. Disponível: https://msdn.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-with-powershell (citado na pág. 64)

[94] Wikipedia, "Postscript," 2017, [accessed 28-March-2017]. [*Online*]. Disponível: https://en.wikipedia.org/wiki/PostScript (citado na pág. 64)

[95] Android Open Source project, "Google play store," 2017, https://developer.android.com/about/dashboards/index.html [accessed 28-March-2017]. [*Online*]. Disponível: https://developer.android.com/about/dashboards/index.html (citado na pág. 65)

[96] Quora, "How many rooted android devices are there?" 2015, https://www.quora.com/How-many-rooted-Android-devices-are-there [Online; accessed 28-March-2017]. (citado na pág. 66)

[97] M. D. Cesaris, "Number of rooted android smartphones," 2014, https://www.linkedin.com/pulse/20140728070440-13998576-number-of-rooted-android-smartphones [Online; accessed 28-March-2017]. (citado na pág. 66)

[98] K. Lucic, "Over 27.44% users root their phone(s) in order to remove built-in apps, are you one of them?" 2014, https://www.androidheadlines.com/2014/11/50-users-root-phones-order-remove-built-apps-one.html [Online; accessed 28-March-2017]. (citado na pág. 66)

[99] A. Boxall, "802015, http://www.businessofapps.com/80-android-phone-owners-china-rooted-device [Online; accessed 28-March-2017]. (citado na pág. 66)

[100] D. Goodin, "10 million android phones infected by all-powerful auto-rooting apps," 2016, https://arstechnica.com/security/2016/07/virulent-auto-rooting-malware-takes-control-of-10-million-android-devices [Online; accessed 28-March-2017]. (citado na pág. 66)

[101] A. Tufts, "One click root," 2012, https://www.oneclickroot.com/reviews/z4root/ [Online; accessed 28-March-2017]. (citado na pág. 71)

Apêndice
Versões do Sistema Operativo *Android*

- ***Android Alpha*** e ***Beta*** **(2007 - 2008)**

Figura 6.1: Icon representativo do *Android* AlphaBeta

A versão Alpha foi utilizada somente pelos componentes da OHA (*Open Handset Alliance*) e era chamada por nomes de robôs, como *Astro Boy*, *Bender* ou *R2-D2*.

Já a versão Beta foi a primeira a ser disponibilizada ao público, tendo um total de 6 versões oficiais publicadas.

- ***Android*** **1.0** ***Astro*** **(2008)**

Figura 6.2: Icon representativo do *Android* Astro 1.0

A primeira versão comercial, foi lançada com o HTC Dream[1] e possuía:

[1]também conhecido como o T-Mobile G1

⇒ *Android Market* (antiga loja de aplicações *Android*).

⇒ Navegador (suportava zoom, formato HTML e XHTML e múltiplas janelas).

⇒ Pastas.

⇒ Acesso à Internet.

⇒ Integração a aplicações do *Google*.

⇒ Reprodução de ficheiros de *mídia*.

⇒ Notificações.

⇒ Ligações por comandos de voz.

⇒ Suporte à câmara (porém sem opções de alterar resolução, cores, etc.), *Wi-fi* e *bluetooth*.

- ***Android* 1.1 – *Battenberg* (2009)**

Figura 6.3: Icon representativo do *Android* Battenberg 1.1

Esta atualização para o *HTC Dream* além de corrigir *bugs* adicionou algumas funcionalidades, como:

⇒ Informações detalhadas e *reviews* na busca por negócios no *Maps*.

⇒ Possibilidade de enviar e guardar anexos em mensagens.

- ***Android* 1.5 – *Cupcake* (2009)**

Figura 6.4: Icon representativo do *Android* Cupcake 1.5

⇒ Possibilidade de uso em sistemas *touchscreen*, com teclado virtual que aceita palavras e dicionários modificados pelo utilizador.

⇒ Suporte a *widgets*.

⇒ Faz filmes em MPEG-4 e 3GP.

⇒ Função copiar e colar no navegador.

⇒ Uso de imagens nos contactos.

⇒ Transições animadas na tela.

⇒ Rotação automática.

⇒ Possibilidade de fazer *upload* de vídeos para o **Youtube** e fotos para o **Picasa**.

- ***Android* 1.6 – *Donut* (2009)**

Figura 6.5: Icon representativo do *Android* Donut 1.6

⇒ Melhorias na pesquisa por voz e entrada de texto para contactos e favoritos.

⇒ Programadores podem incluir as suas criações nas buscas.

⇒ Motor de fala que permite às *apps* falarem uma sequência de texto.

⇒ Melhorias nos resultados do *Android Market*.

⇒ Velocidade e integração entre a câmara para fotos e câmara para vídeo.

⇒ Utilizadores podem selecionar fotos para exclusão.

⇒ Conexão *CDMA/EVDO, Wi-fi- 802.1x* e *Vpn,'s*.

⇒ Suporte para telas WVGA.

- ***Android* 2.0 (2.0.1 e 2.1) – *Éclair* (2009 - 2010)**

Figura 6.6: Icon representativo do *Android* Éclair 2.0

⇒ Possibilidade de adicionar várias contas ao dispositivo e sincronizá-las.

⇒ Suporte ao *e-mail Microsoft Exchange*.

⇒ *Bluetooth* 2.1.

⇒ Melhorias na interação com os contactos.

⇒ Pesquisar as mensagens SMS e MMS armazenadas.

⇒ Novos recursos de câmara, como flash, zoom digital, efeitos de cor, etc.

⇒ Melhorias de velocidade e dicionário inteligente no teclado virtual.

⇒ Suporte a HTML 5.

⇒ Suporte a novos tamanhos de tela e resoluções.

⇒ *WallPapers* animados.

- ***Android* 2.2 (2.2.1; 2.2.2 e 2.2.3) – *Froyo* (2010)**

Figura 6.7: Icon representativo do *Android* Froyo 2.2

⇒ Otimizações de memória, desempenho e velocidade.

⇒ Motor de *JavaScript* no navegador.

⇒ Suporte ao *Android Cloud Computing*.

⇒ Melhorias no suporte ao *Microsoft Exchange*.

⇒ Possibilidade de ser usado como *hotspot Wi-fi*.

⇒ Opção para desativar os dados através da rede móvel.

⇒ Atualizações no *Android Market*.

⇒ Troca rápida entre idiomas e dicionários na digitação.

⇒ Compatibilidade do *bluetooth* com veículos e *docks* (Interface gráfico de utilizador).

⇒ Suporte a senhas numéricas e alfanuméricas.

⇒ Suporte a *upload* na navegação.

⇒ Suporte a *GIF* no navegador.

⇒ Suporte à instalação de aplicações na memória externa, como cartões de memória.

⇒ Suporte para telas HD 720p de até 4' (aproximadamente 10,16 cm) e até 320 DPI (pontos por polegada).

- ***Android* 2.3 (2.3.1; 2.3.2) – *Gingerbread* (2010 - 2011)**

Figura 6.8: Icon representativo do *Android* Gingerbread 2.3

⇒ Design de interface de utilizador atualizado com maior simplicidade e rapidez.

⇒ Suporte para telas e resoluções extra-grande.

⇒ Suporte nativo de protocolos de telefonia via Internet SIP e VoIP.

⇒ Suporte à tecnologia NFC.

⇒ Novos efeitos de áudio, como *reverb*, *equalizing*, virtualização de *headphones* e *bass boost*.

⇒ Novo gestor de *downloads*, dando aos utilizadores fácil acesso a qualquer ficheiro descarregado a partir do *browser*, *e-mail* ou outra aplicação.

⇒ Suporte para múltiplas câmaras no dispositivo.

⇒ Suporte para reprodução de vídeo *WebM/VP8*, e codificação de áudio AAC.

⇒ Melhor gestão de energia.

⇒ Melhorias para os programadores de jogos.

⇒ Suporte nativo para mais sensores, como giroscópio e barómetro.

- ***Android* 2.3.3 (2.3.4; 2.3.5; 2.3.6 e 2.3.7) – *Gingerbread* (2011)**

  ⇒ Suporte para voz ou *chat* de vídeo usando o *Google Talk*.

  ⇒ Suporte à conexão com um periférico USB com *software* compatível e uma aplicação compatível no dispositivo.

  ⇒ Mudança na criptografia padrão para *SSL* de *AES256-SHA* para *RC4-MD5*.

  ⇒ Melhorias de *software* da câmara.

  ⇒ Melhoria da eficiência da bateria.

  ⇒ Suporte do *Google Wallet*.

- ***Android* 3.0 – *Honeycomb* (2011)**

Figura 6.9: Icon representativo do *Android* Honeycomb 3.0

⇒ Nova interface de utilizador "Holográfica". Otimizada para o uso em *tablets*.

⇒ Adicionada a barra de acesso rápido a notificações, status e botões de navegação na parte inferior da tela.

⇒ Adicionada a barra de ação, que dá acesso a opções contextuais, navegação, *widgets*, ou outros tipos de conteúdo na parte superior da tela.

⇒ Multi-tarefa simplificada que exibe as aplicações recentes e permite aos utilizadores trocar rapidamente de uma aplicação para outra.

⇒ Teclado redesenhado, tornando a digitação mais rápida, eficiente e precisa em tamanhos de tela maior.

⇒ As várias janelas do navegador foram agrupadas em abas, além de preenchimento automático e um novo modo de navegação anónima.

⇒ Acesso rápido à câmara e seus recursos.

⇒ A aceleração do *hardware*.

⇒ O suporte para processadores *multi-core*.

⇒ Capacidade de encriptar todos os dados do utilizador.

⇒ *HTTPS* melhorado com *SNI*.

- ***Android* 3.1 – *Honeycomb* (2011)**

  ⇒ Melhorias na interface do utilizador.

  ⇒ Conectividade para acessórios USB (*USB On-The-Go*).

  ⇒ Suporte para teclados externos e dispositivos apontadores (*lasers*).

  ⇒ Suporte para *joysticks* e *gamepads*.

  ⇒ Suporte para reprodução de áudio *FLAC*.

  ⇒ Suporte para *proxy HTTP* para cada ponto de acesso *Wi-Fi* conectado.

- ***Android* 3.2 (3.2.1; 3.2.2; 3.2.3; 3.2.4; 3.2.5 e 3.2.6) – *Honeycomb* (2011 - 2012)**

  ⇒ Suporte de *hardware* melhorado, incluindo otimizações para uma ampla gama de *tablets*.

  ⇒ Melhoria de sincronização e acesso aos ficheiros no cartão SD.

  ⇒ Modo de exibição de compatibilidade para aplicações que não foram otimizadas para *tablet*.

  ⇒ Novas funções de suporte de exibição aos programadores.

  ⇒ Correções de *bugs*, segurança, estabilidade e melhorias *Wi-Fi*.

  ⇒ Melhor suporte *Adobe Flash* no navegador.

  ⇒ Suporte ao "*Pay as You Go*".

- ***Android* 4.0 (4.0.1; 4.0.2; 4.0.3 e 4.0.4) - *Ice Cream Sandwich* (2011 - 2012)**

Figura 6.10: Icon representativo do *Android* Ice Cream Sandwich 4.0

⇒ Aperfeiçoamentos da interface *Holo*.

⇒ Pastas *drag-and-drop*.

⇒ Captura de tela integrada através dos botões de energia e de volume.

⇒ Melhoria da correção de erros no teclado.

⇒ Capacidade de aceder aplicações diretamente a partir da tela de bloqueio.

⇒ Melhor integração de voz, na fala em tempo real no texto falado.

⇒ Face *Unlock*, um recurso que permite aos utilizadores desbloquear os aparelhos usando o *software* de reconhecimento facial.

⇒ Sincronização automática do *Chrome* com os favoritos dos utilizadores.

⇒ Possibilidade de definir um limite de dados que irá gerar avisos quando se aproximar e permite desativar os dados móveis quando esse limite é excedido.

⇒ Capacidade de desligar todos as aplicações recentes de uma só vez.

⇒ Melhoria da câmara como *lag* zero, obturador, configurações de intervalos de tempo, modo panorama e a capacidade de zoom durante a gravação.

⇒ Editor de fotos nativo.

⇒ *Android Beam*, recurso de comunicação de baixo alcance que permite a troca rápida de favoritos, informações de contacto, direções, vídeos do *YouTube* e outros dados.

⇒ Suporte para o formato de imagem *WebP*.

⇒ A aceleração de *hardware* da interface de utilizador.

⇒ *Wi-Fi Direct*.

⇒ Gravação de vídeo 1080p, estabilização de vídeo e resolução *QVGA*.

⇒ *Android VPN Framework* (AVF), e TUN.

⇒ Melhorias para gráficos, bases de dados, correção ortográfica e funcionalidade *Bluetooth*.

- ***Android* 4.1 (4.1.1 e 4.1.2) – *Jelly Bean* (2012)**

Figura 6.11: Icon representativo do *Android* Jelly Bean 4.1

⇒ Sincronismo *Vsync* em todos desenhos e animações feitos pela estrutura do *Android*.

⇒ *Buffer* triplo para gráficos.

⇒ Capacidade de desativar as notificações numa base específica da aplicação.

⇒ Atalhos e *widgets* podem ser automaticamente reparados ou refeitos sob medida para permitir a novos itens caberem na *home*.

⇒ Transferência de dados *Bluetooth* para *Android Beam*.

⇒ *Tablets* com telas menores agora podem usar uma versão estendida da interface e tela dos telefones.

⇒ Melhoria da aplicação da câmara.

⇒ Áudio multi-canal.

⇒ O *codec Fraunhofer FDK AAC* torna-se padrão no *Android*, acrescentando *AAC 5.1* para os canais de codificação/descodificação.

⇒ Áudio USB com conversores digital-analógico.

⇒ Encadeamento de áudio (reprodução contínua).

⇒ Possibilidade de expandir/contrair notificações com gestos de apenas um dedo.

- ***Android* 4.2 (4.2.1 e 4.2.2) – *Jelly Bean* (2012)**

⇒ Melhorias na tela de bloqueio, incluindo suporte a *widget* e a capacidade de acesso direto à câmara.

⇒ Configurações rápidas.

⇒ Protetor de tela "*Daydream*", mostrando informações quando está disponível ou conectado.

⇒ Possibilidade de múltiplas contas de utilizador (para *tablets*).

⇒ Reestruturação do *Bluetooth*, que permite um melhor suporte para múltiplos monitores e *displays wireless*.

⇒ Melhorias de acessibilidade, como tocar três vezes para ampliar a tela inteira, modo panorâmico e zoom com dois dedos.

⇒ Saída de voz e navegação via gesto para deficientes visuais.

⇒ Novo relógio com horários mundiais, cronometro e *timer*.

⇒ Aumento do número de notificações e de recursos, permitindo que os utilizadores respondam a certas notificações sem lançar a aplicação.

⇒ *SELinux* (módulo de segurança).

⇒ Mensagens em grupo.

⇒ Suporte a *gamepads* e *joysticks Bluetooth HID*.

⇒ Toque longo para ligar e desligar funções, tocando nos ícones *Wi-Fi* e *Bluetooth*.

⇒ Novas notificações de *download*, que agora mostram a percentagem estimada e tempo restante dos mesmos.

⇒ Novos sons para o carregamento sem fios e bateria fraca.

⇒ Novo modo de depuração *USB* (*whitelist*).

- ***Android* 4.3 (4.3.1) – *Jelly Bean* (2013)**

⇒ Possibilidade de usar o *Bluetooth* com baixa energia.

⇒ Suporte a controlo remoto de *Bluetooth Áudio/Vídeo* (AVRCP).

⇒ Suporte a *OpenGL ES 3.0*, permitindo melhores gráficos de jogos.

⇒ Modo de acesso restrito para novos perfis de utilizador.

⇒ Recursos de auto-completar na marcação de números de telefone.

⇒ Melhorias para o *Photo Sphere*.

⇒ Reformulada a interface da câmara.

⇒ Suporte à resolução 4K.

⇒ Identificação de redes *Wi-Fi* até mesmo quando o *Wi-Fi* estiver desligado.

- ***Android* 4.4 (4.4.1; 4.4.2; 4.4.3 e 4.4.4) – *KitKat* (2013 - 2014)**

Figura 6.12: Icon representativo do *Android* KitKat 4.4

⇒ Interface repensada com elementos brancos ao invés de azul.

⇒ Restrição às aplicações ao aceder ao armazenamento externo, com exceção das suas próprias diretorias.

⇒ Otimizações para o desempenho em dispositivos com características inferiores.

⇒ Capacidade de impressão sem fios.

⇒ Emulação de cartões NFC, possibilitando ao dispositivo substituir os *Smart cards*.

⇒ Novo seletor de ficheiros que permite aos utilizadores aceder a ficheiros de várias fontes (incluindo aqueles expostos por aplicações, tais como serviços de armazenamento *on-line*).

⇒ Melhorias de áudio, como monitorização e potencializador de volume máximo.

⇒ Recurso nativo de gravação de tela.

⇒ *Infrared Blaster* nativo (recurso que permite usar o *smartphone* como um controle remoto).

⇒ Mais opções de acessibilidade, como estatísticas de bateria.

⇒ *Android Runtime*(ART), novo ambiente de execução experimental, substituindo a máquina virtual *Dalvik*.

⇒ Suporte a *MAP Message Access Bluetooth Profile*.

⇒ Melhorias para a câmara, como foco automático, balanço de branco e HDR +.

⇒ A aplicação da câmara agora carrega Fotos do *Google+*, em vez de *Gallery*.

⇒ Diversos aprimoramentos e correções na navegação, como HTML 5.

- ***Android* 4.4W (4.4W.1 e 4.4W.2) – *KitKat* (2014)**

  ⇒ Primeiro lançamento voltado para os *Wereables* (vestíveis).

  ⇒ Atualizações e atualizações do *Maps*.

  ⇒ Suporte ao *GPS*.

  ⇒ Reprodução de música *offline*.

- ***Android* 5.0 (5.0.1 e 5.0.2) – *Lollipop* (2014)**

Figura 6.13: Icon representativo do *Android* Lollipop 5.0

  ⇒ Suporte para *CPUs* de 64 bits.

  ⇒ *OpenGL ES* 3.1.

  ⇒ Gráficos vetoriais, que melhoram a definição de imagens.

  ⇒ Pré-visualização de impressão.

  ⇒ Nova interface de utilizador chamada *Material Design*.

  ⇒ Bandeja de notificações revigorada e configurações rápidas *pull-down*.

  ⇒ Tecnologia "Volta", para melhorias de vida da bateria.

  ⇒ As pesquisas podem ser feitas dentro das configurações do sistema para um acesso mais rápido às configurações específicas.

  ⇒ A Tela de bloqueio fornece atalhos para as configurações da aplicação e notificações.

  ⇒ *Logins* e contas de utilizador disponíveis em mais dispositivos.

  ⇒ Entrada e saída de áudio através de dispositivos USB.

  ⇒ Regresso da habilidade das aplicações de terceiros, como ler e modificar os dados localizados em qualquer ponto de armazenamento externo, como os cartões SD.

⇒ Aplicações usadas recentemente são lembradas mesmo depois de reiniciar o dispositivo.

⇒ Recurso *Tap and Go* permite que os utilizadores migrem rapidamente para um novo dispositivo *Android*, utilizando *NFC* e *Bluetooth* para transferir dados da sua conta do *Google*, configurações, dados de utilizador e aplicações instaladas.

⇒ Lanterna nativa.

⇒ Possibilidade de personalizar as notificações das aplicações.

⇒ Função de bloqueio inteligente.

- ***Android* 5.1 (5.1.1) – *Lollipop* (2015)**

⇒ Capacidade de conectar a redes *Wi-Fi* e dispositivos de *Bluetooth* via configurações rápidas.

⇒ Suporte para múltiplos cartões SIM.

⇒ Proteção do aparelho: se um dispositivo for perdido ou roubado ele permanecerá bloqueado até que o proprietário conecte a sua conta *Google*, mesmo se o dispositivo é redefinido para as configurações de fábrica.

⇒ Chamadas de voz de alta definição disponível entre dispositivos compatíveis com o *Android* 5.1.

⇒ Melhorias no sistema de notificação de prioridade, para se aproximar ao modo silencioso que foi removido no *Android* 5.0.

- ***Android* 6.0 - *Marshmallow* (2015)**

Figura 6.14: Icon representativo do *Android* Marshmallow 6.0

⇒ *Now on tap:* recurso que contextualiza o *Google Now* nas aplicações com um toque no botão *home*.

⇒ Modo *Doze*: recurso que economiza a bateria do dispositivo automaticamente quando está em *stand-by*.

⇒ Gaveta de aplicações na vertical e com procura por ordem alfabética.

⇒ Suporte nativo para leitores de impressão digital.

⇒ Modo "Não Perturbe".

⇒ Suporte para USB *Type-C*.

⇒ *Backup* e restauração automática no "Disco"para dados e aplicações.

⇒ Modo de tela 4K para *apps*.

⇒ Adaptação de memória externa (*SD cards*) como parte da memória interna.

⇒ Suporte para *MIDI* em instrumentos musicais.

- ***Android* 7.0 - *Nougat*[2] (2016)**

Figura 6.15: Icon representativo do *Android* Nougat 7.0

⇒ Encriptação nativa: recurso que trará uma camada extra de segurança para os utilizadores do sistema.

⇒ Novos *emojis*: O *Android Nougat* conta com 72 novos *emojis*, que variam entre alimentos e a representatividade feminina.

⇒ Controlar o nível de importância das notificações manualmente: Será possível organizar as notificações em seis níveis diferentes. São eles: Bloqueado, Mínimo, Baixo, Normal, Alto, Urgente.

⇒ Modo de realidade virtual: A nova versão do SO do *Google* deve contar com suporte para realidade virtual. No menu do novo sistema encontra-se uma tela à espera para ser preenchida com uma lista de aplicações que utilizem uma API compatível com *apps* de RV.

⇒ Responder mensagens pela barra de notificações: Agora as mensagens poderão ser respondidas a partir da tela de bloqueio, pela própria notificação, sem a necessidade de sair da aplicação, para responder a alguma mensagem.

[2] foi lançada este ano (2016) durante o Google I/O 2016 e por enquanto ainda não está disponível para os utilizadores, apenas para programadores

⇒ Melhorias no *Launcher Google Now*: O recurso passou por melhorias. Uma delas é que agora é possível utilizar o gesto de pinças (juntando o dedo indicador com o polegar como se fosse beliscar a tela) na tela inicial para voltar a tela de visualização geral, onde, no fim da página, irão aparecer *wallpapers*, *widgets* e configurações.

⇒ Capacidade de alterar o tamanho e a interface da fonte.

⇒ Integração do *Mono Play* para deficientes auditivos.

⇒ *API JobScheduler* torna o *smartphone* mais rápido.

⇒ Nova função de multi-janela.

⇒ Novo painel de notificações.

⇒ Introdução das *apps* instantâneas.

⇒ Nova API de renderizações, a *Vulkan 3D*.

⇒ Modo *Doze* aprimorado; Anteriormente o recurso de economia de bateria só funcionava quando o telefone estava em repouso. Com o *Android Nougat*, o *smartphone* economizará bateria sempre que a tela se desligar.